REASONING MODELED AS A SOCIETY OF COMMUNICATING EXPERTS

LUC STEELS

June,1979

MASSACHUSETTS INSTITUTE OF TECHNOLOGY

ARTIFICIAL INTELLIGENCE LABORATORY

ABSTRACT

This report describes a domain independent reasoning system. The system uses a frame-based knowledge representation language and various reasoning techniques including constraint propagation, progressive refinement, natural deduction and explicit control of reasoning. A computational architecture based on active objects which operate by exchanging messages is developed and it is shown how this architecture supports reasoning activity. The user interacts with the system by specifying frames and by giving descriptions defining the problem situation. The system uses its reasoning capacity to build up a model of the problem situation from which a solution can interactively be extracted. Examples are discussed from a variety of domains, including electronic circuits, mechanical devices and music.

The main thesis is that a reasoning system is best viewed as a parallel system whose control and data are distributed over a large network of processors that interact by exchanging messages. Such a system will be metaphorically described as a society of communicating experts.

CONTENTS

PREFACE

We have designed and implemented a domain independent *reasoning system.* This system, known as the XPRT-system, accepts definitions of concepts and consults these definitions when it is asked to solve particular problems.

Every attempt to construct or study reasoning systems has to address at least three questions:
(i) What is the architecture of a reasoning system?
(ii) How should knowledge be represented and organized?
(iii) How should knowledge be activated? and
(iv) How do we model complex problem solving?

Concerning the architecture problem we will propose that a reasoning system is best implemented as a parallel system whose control and data are distributed over a large network of processors that interact by exchanging messages. Such a system will be metaphorically described as *a society of communicating experts.*
The fundamental unit of the system is an active, independently operating object, called an expert, which contains a body of knowledge about a certain (limited) subject-matter, and a script specifying how the expert should respond to messages from other experts. A group of experts that have the same task-structure or whose subject-matters are part of the same domain will be called a society. Reasoning behavior will be studied in terms of patterns of messages that are exchanged between experts of such a society.

Concerning the problem of representing knowledge, we will develop a theory based on the notion of *a frame.* A frame is a collection of important questions that should be asked about a limited subject-matter. A frame is the basis for constructing descriptions. A description is a way to refer to individuals by specifying what role they play in a frame, i.e. to what question they are an answer.
A frame contains partial answers to its questions in the form of descriptions which are always true, descriptions which characterize the range of an answer, methods to find the answer, etc.

Concerning the problem of activating these frames, we will propose a concrete reasoning system based on the metaphor of a question-answering activity. Because a frame is viewed as a collection of important questions, reasoning about the subject-matter of the frame is viewed as finding the answer to the various questions posed by the frames that become active in a given problem situation. Some of these answers may come from the initial specifications of the problem, others may come from the descriptions attached to the frames.
This question answering activity is mechanized in terms of the society of experts metaphor: An expert is created for each question posed by a frame. These experts start out with the answers available in the frame. By communicating with other experts they try to accumulate more information thereby progressively refining the answer to the question they are responsible for.

Sophisticated problem solving is obtained by viewing a whole society of experts as one

single expert. This expert tries to solve complicated problems by interaction with other (complex) experts. For example there could be different experts for each point of view from which the problem can be approached. Or there could be pairs of experts where one proposes solutions and the other criticizes them. Complex problem solving is therefore also approached in terms of the society of experts metaphor.

CONTRIBUTIONS

We believe that this work makes a number of important contributions to the state of the art in reasoning and knowledge representation. Here are some of the most important ones of these contributions:

(i) We have been able to construct a synthesis of work on procedural deduction and work on knowledge representation languages. Our system has all the capabilities of a procedural deduction system but accepts knowledge in the form of natural-language like descriptions. From this perspective the present work is an important step towards systems that are able to learn by being told (instead of programmed).

(ii) We have been able to define a system based on the idea of a network of independently operating computational objects that perform reasoning by communicating messages. The importance of this result is twofold. On the one hand it is a giant leap forward in the amount of reasoning that can be done given a certain stretch of real time. This might prove crucial in the construction of intelligent systems which have to perform in real time. On the other hand a parallel systems architecture proves to be more satisfactory in modelling the human brain because this is a parallel system itself.

(iii) We have explored a synthesis of declarative and procedural ways of representing knowledge, not by viewing everything as a procedure (as is done in the 'procedural embedding of knowledge paradigm' of the ACTOR-school), or by requiring that procedures are attached in order to activate the frames (as is done in other frame-systems like FRL) but by viewing everything as a description and by viewing the activation problem as a reasoning problem. The importance of this result is that we obtain a 'cleaner' system of knowledge representation and that the user has to specify less information than used to be the case. It will be shown that we nevertheless retain sufficient control over the reasoning processes by applying the explicit control of reasoning paradigm.

(iv) We have constructed an efficient reasoner. The time required to perform a deduction does not significantly increase with the amount of knowledge available or the number of facts already deduced. This is so because knowledge is structured in a way that rules are selectively activated based on their relevance to the subject-matter being reasoned about and that facts are not stored in a global database but organized in units based on their subject-matter thus enabling lookup based on content rather than form. Further efficiency is obtained by using powerful heuristics based on criteriality and projectivity properties of concepts in order to find the referents of descriptions.

HISTORICAL REMARKS

This document reflects the author's state of research in May 1979. Many issues have been explored since then and there is no doubt that there are many more issues remaining. This work should therefore be seen as the first step on a path towards a

system that is able to learn by being told knowledge in a natural-language like description language. The major issue we have tackled since finishing this work is to represent and reason about prototypes. Prototypes are our solution to certain problems of quantification, representation of sets, nonmonotonic reasoning, a.o.. The introduction of prototypes has lead us to reorganize substantially the present system: we have introduced physical sharing of descriptions instead of copying as is done here, we have introduced experts for prototypes in addition to experts for objects in a particular problem situation. We have also introduced a new internal representation of descriptions making the network of processors that constitute a society more like a semantic network. Each node in this network is an active object. Travel from one node to another can occur in any direction and scenarios for accomplishing cognitive goals (such as finding the most specific concept given a collection of concepts, finding whether a certain object is described in terms of a certain description, etc) are executed in parallel. These new developments will be documented in papers now in preparation.

Although much further work is needed, the framework sketched in this document proved to be a sound base for exploring more advanced issues of cognition. The fact that several serious applications are now in the planning stages is an additional source of satisfaction.

STRUCTURE OF THE DOCUMENT

The text is organized in a modular fashion and can therefore be read on several levels of detail. The first section of every chapter is devoted to presenting and justifying a series of principles. A principle is a fundamental hypothesis. The justification of a principle will be in the form of a functional argument: we will try to show why a certain hypothesis is plausible or necessary given the task-structure of a reasoning system.
The second section of every chapter is devoted to introducing a conceptual framework, a concrete system based on this framework and a number of examples from a variety of domains. If possible we try to use examples that have been used in the literature to make comparisons easy.

Before we start the main text, an overview of the capacities of the reasoning system is given in an introductory chapter. This overview should give the reader an intuitive idea of what is possible with the system as it has been developed so far.

ACKNOWLEDGEMENT

The work described in this document was done at the MIT Artificial Intelligence Laboratory based on an ITT-fellowship. I am very much indebted to many members of that laboratory for the many discussions and the comments on this work. I especially thank those who read preliminary drafts of this work: Carl Hewitt, Marvin Minsky, Henry Lieberman, Jon Doyle, Roger Duffy, Ken Kahn, Bill Kornfeld, David Levitt, Jerry Barber.

I also thank my wife Dora for her help and support.

INTRODUCTION

1. INTRODUCTION

In this first chapter we will give an intuitive idea of the reasoning capabilities we have been able to circumscribe. We will do this by interacting with the current implementation of the reasoner. The reader should not worry if something in this demonstration is not understood. All will be explained in much greater detail later on.

We will concentrate on two examples. One from the domain of family relations and one from the domain of physical devices, in particular electronic circuits. We start by introducing some concepts for family relationships, such as parent, person, mother, etc. A concept has aspects for the important questions to be asked about the situation to which the concept applies. Thus we define that the concept for parent has a child-aspect by writing

```
(PARENT
   (WITH CHILD))
```

Each concept has a SELF-aspect. This is an aspect pointing to the object to which the concept applies. Thus we say

```
(PARENT
   (WITH SELF)
   (WITH CHILD))
```

or

```
(PERSON
   (WITH SELF)
   (WITH BIRTHDAY) ...).
```

A description is a way to talk about an instance of a concept. An instance of the PERSON concept is represented as

```
(A PERSON)
```

Or an instance of the PARENT concept is represented as

```
(A PARENT).
```

Some concepts are special in that there is only one possible object that can ever fill its self-aspect. Such concepts will be called individual-concepts. If the concept is an individual concept we do not write the indefinite article in a description using this concept. For example we just say

```
JOHN
```

if we want to talk about the instance of the individual concept JOHN (because there is only one JOHN anyway).

We can also talk about an object as playing a particular role in an instance of a concept. This role is one of the aspects of the concept and will be called the view of the description. For example one can refer to an object as a child of a parent by writing

```
(A CHILD OF A PARENT).
```

'Child' is the aspect that is the view here. If we know that there is only one instance satisfying a certain description we will use the definite article as in

```
(THE CHILD OF A PARENT)
```

The aspects of an instance of a concept are 'filled' by particular individuals. To indicate which individual is filling an aspect in a particular instance we will attach descriptions describing this individual. Attachment is represented by writing the description after the aspect-name. Suppose for example that we want to talk about an object as a parent whose child is John. Then we can do so by saying

```
(A PARENT
   (WITH CHILD JOHN))
```

A description can also be further constrained by giving a description of the instance itself. For example if we want to introduce an object by saying that it is the child of a parent who is Mary, then we can do this as follows:

```
(THE CHILD OF A PARENT
      (WHO IS MARY))
```

The description (WHO IS MARY) restricts the scope of the parent in this description.

It is possible to introduce constraints on every instance or on the objects filling the aspect of every possible instance of a concept by attaching descriptions to the aspects in the definition. For example, if we want to say that every female-person is a person, then we can do so as follows:

```
(FEMALE-PERSON
   (WITH SELF
         (A PERSON)))
```

Attachment now works like implication: every time an instance of a female-person is considered, the reasoner will deduce that this instance is a person.

A *frame* is a concept with a series of aspects and with descriptions attached to these aspects representing constraints on what objects can play the roles indicated by the aspects.

Here is another example. Suppose we want to say that every mother is a parent, then we can do this by attaching a description to the self-aspect of the mother frame:

```
(MOTHER
  (WITH SELF
        (A PARENT))
  (WITH CHILD))
```

Now suppose we want to be more specific and say that every mother is a parent such that the child of the mother is the child of the parent. This is done by introducing co-referential-links in a frame. Co-referential links are represented by assigning a (local) name to the object filling the aspect in the instance of the frame under consideration and repeating this assignment at every aspect that is co-referentially linked.

For example, we need co-referential links between the child-aspect of mother and the child-aspect of the description making reference to the parent-concept. Let us use the name 'THE-CHILD', then we can say

INTRODUCTION

```
(MOTHER
  (WITH SELF
        (A PARENT
           (WITH CHILD (= THE-CHILD))))
  (WITH CHILD (= THE-CHILD)))
```

This frame now contains the information that every mother is a parent whose child is the child of the mother.

It is allowed to use a conjunction of descriptions as in:

```
(MOTHER
  (WITH SELF
        (AND (A FEMALE-PERSON)
             (A PARENT
                (WITH CHILD (= THE-CHILD)))))
  (WITH CHILD
        (= THE-CHILD)))
```

which says that every mother is a female-person AND a parent whose child is the child of the mother. Other connectives like OR, XOR and NOT can be used as well.

Let us now feed these frames to the reasoner and start up a little discussion. The reasoner prompts with a >>-sign. Input is in small letters (except for the frames). Output is in capital letters.

First we say that Mary is a mother whose child is Joan:

```
>> Mary is (a mother (with child Joan))
```

The reasoner has stored this fact and has deduced all consequences of the situation introduced by this fact. We can find out about this by asking questions. First we check whether the initial description is there:

```
>> Is Mary (a mother (with child Joan))?
YES,
MARY IS (A MOTHER (WITH CHILD JOAN))
```

We can also perform queries based on partial descriptions:

```
>> Is Mary (a mother)?
YES,
MARY IS (A MOTHER (WITH CHILD JOAN))
```

We can ask informative questions:

```
>> Who is (a mother (with child Joan))?
MARY IS (A MOTHER (WITH CHILD JOAN))
```

And it is of course possible that the answer is not known:

```
>> Is Joan (a mother)?
I DON'T KNOW
```

The reasoner knows also about the child:

```
>> Is Joan (a child of a mother (who is Mary))?
YES,
JOAN IS (A CHILD OF A MOTHER (WHO IS MARY))
```

And it knows all implications of the fact that Mary is a mother. One of these implications is that she is a parent:

INTRODUCTION

```
>> Is Mary (a parent)?
YES,
MARY IS (A PARENT (WITH CHILD JOAN))
```

Another implication is that Mary is a female-person and therefore a person:

```
>> Is Mary (a person)?
YES,
MARY IS (A PERSON)
```

Now we illustrate another capability of the reasoner: the ability to justify its beliefs. We start by asking why the last statement is believed:

```
>> Why?
MARY IS (A PERSON)
BECAUSE
MARY IS (A FEMALE-PERSON)
```

We can continue asking questions:

```
>> Why?
MARY IS (A FEMALE-PERSON)
BECAUSE
MARY IS (AND (A FEMALE-PERSON)
             (A PARENT (WITH CHILD JOAN)))
>> Why?
MARY IS (AND (A FEMALE-PERSON)
             (A PARENT (WITH CHILD JOAN)))
BECAUSE
MARY IS (A MOTHER (WITH CHILD JOAN))
>> Why?
MARY IS (A MOTHER (WITH CHILD JOAN))
BECAUSE
  YOU TOLD ME SO
```

This is of course the ultimate justification.

We can perform much smarter forms of reasoning: the reasoner has complicated methods for finding the referent of a descriptions, it can deal with partial descriptions, conditional descriptions, anonymous objects, method frames, etc. All of this will be discussed more extensively later on. One of the most important properties of the system is that it is completely domain-independent and accepts at any time definitions of new concepts. It is therefore more like a learning system, learning in the sense of having a teacher (or a dictionary for that matter) that explains new concepts and their meanings. For the domain of family-relations we could therefore introduce concepts for family, uncle, brother, etc., and then feed whole family trees to the reasoner and perform all the queries we want to.

By introducing the right concepts, we can talk about any domain. To illustrate that we will look at many domains in this document. Right now we will look at one other domain: physical devices. In particular, we will redo the example of circuit analysis discussed in Sussman and Steele (1978). One thing you are allowed to do is attach (LISP) procedures instead of descriptions to an aspect in a frame. The result of the procedure is then one of the constraints on what object fills that aspect. For example, we can introduce a frame for SUM with aspects for the addend and the augend. This frame acts like a constraint among the value of a set of objects. The procedures attached to the various slots make sure that whenever sufficient information is there all values are computed.

INTRODUCTION

```
(SUM
  (WITH SELF
        (PLUS (= THE-ADDEND) (= THE-AUGEND)))
  (WITH ADDEND
        (DIFFERENCE (= THE-SELF) (= THE-AUGEND)))
  (WITH AUGEND
        (DIFFERENCE (= THE-SELF) (= THE-ADDEND))))
```

So if we describe something as

```
   (A SUM
         (WITH ADDEND 5)
         (WITH AUGEND 10))
```

then this means the same as describing it as 15.

Here is a frame for product:

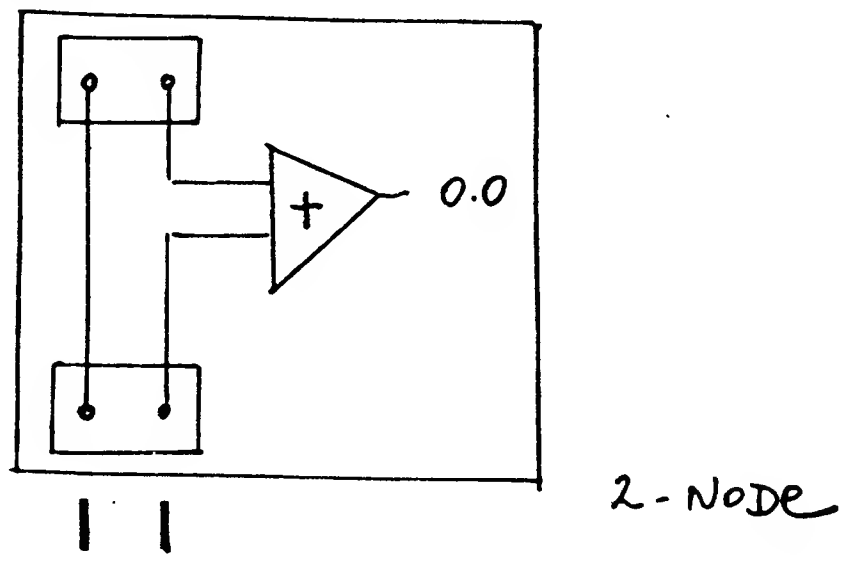
```
(PRODUCT
  (WITH SELF
        (TIMES (= THE-MULTIPLICAND)(= THE-MULTIPLIER)))
  (WITH MULTIPLICAND
        (QUOTIENT (= THE-SELF)(= THE-MULTIPLIER)))
  (WITH MULTIPLIER
        (QUOTIENT (= THE-SELF) (= THE-MULTIPLICAND)))).
```

PLUS, DIFFERENCE, TIMES, and QUOTIENT are LISP-functions.

We will use these frames in studying a simple circuit. We start by introducing some frames for components of circuits. First comes a frame for a terminal. A terminal has a potential (represented by the voltage-aspect) and a current:

```
(TERMINAL
  (WITH SELF)
  (WITH VOLTAGE)
  (WITH CURRENT))
```

We can put two terminals together into a two-terminal node or 2-node. In a 2-node the potentials of the two terminals are equal and the sum of the currents has to be 0.0. Graphically this is represented as

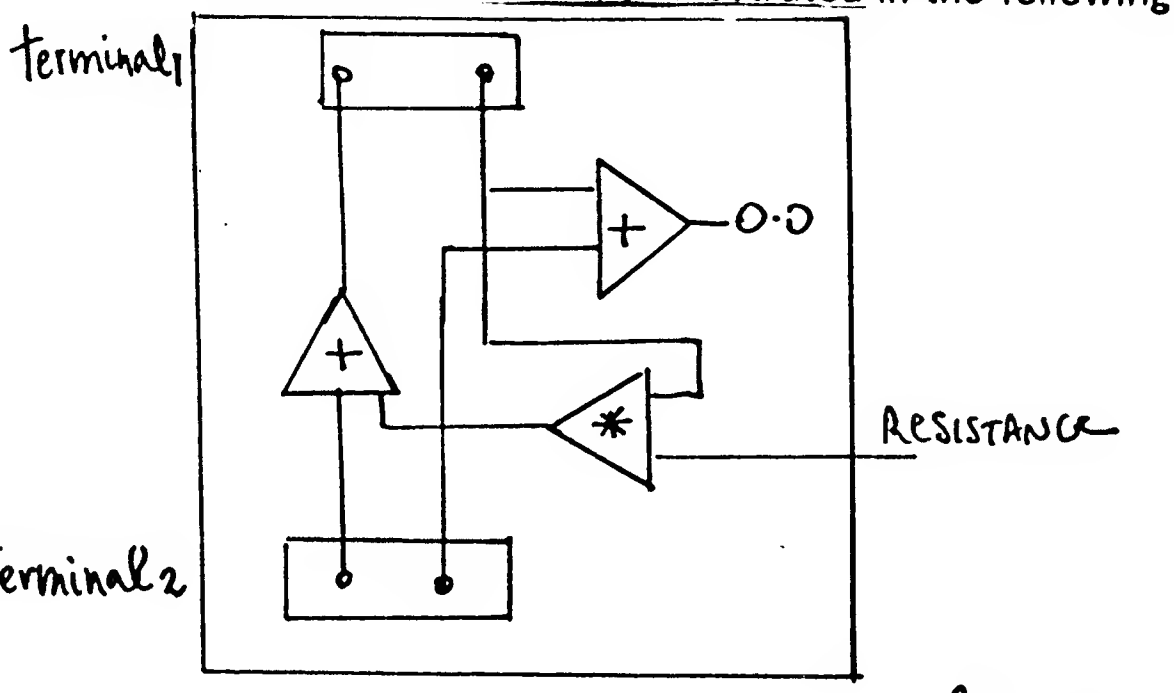


This can be expressed in descriptions by saying that a 2-node is an object with aspects for two terminals: terminal1 and terminal2. The terminal2 is described as a terminal whose voltage is equal to the voltage of the terminal1 and whose current is the augend of a sum whose result is 0.0 and whose addend is the current of terminal1:

INTRODUCTION

```
(2-NODE
 (WITH SELF)
 (WITH TERMINAL1)
 (WITH TERMINAL2
       (A TERMINAL
          (WITH VOLTAGE
                (THE VOLTAGE OF A TERMINAL
                     (WHICH IS (= THE-TERMINAL1))))
          (WITH CURRENT
                (THE AUGEND OF A SUM
                     (WHICH IS 0.0)
                     (WITH ADDEND
                           (THE CURRENT OF A TERMINAL
                                (WHICH IS (= THE-TERMINAL1)))))))))
```

A resistor involves two terminals and a resistance which are related via two SUM-constraints and a PRODUCT-constraint as illustrated in the following diagram:



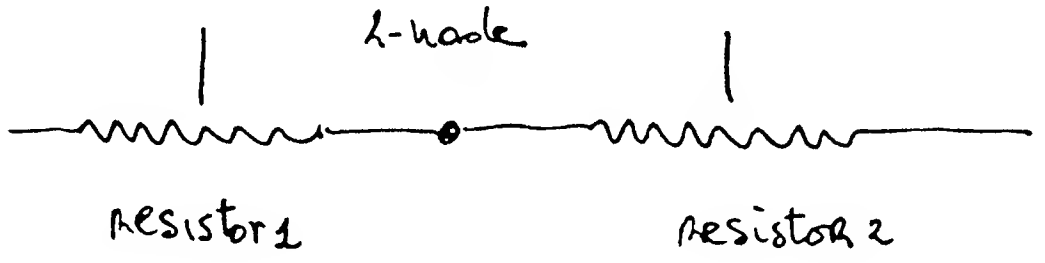
RESISTOR

The resistor-frame itself has aspects for the resistor itself, two terminals called terminal1 and terminal2 and a resistance. The current of the first terminal is described as the augend of a sum whose result is 0.0 and whose addend is the current of the other terminal. The voltage of the first terminal is described as a sum with the voltage of the other terminal as addend and the product of the resistance and the current of the first terminal as augend.

INTRODUCTION

```
(RESISTOR
 (WITH SELF)
 (WITH TERMINAL2)
 (WITH RESISTANCE)
 (WITH TERMINAL1
       (A TERMINAL
          (WITH CURRENT (= THE-CURRENT-OF-TERMINAL1)
                (THE AUGEND OF A SUM
                     (WHICH IS 0.0)
                     (WITH ADDEND
                           (THE CURRENT OF A TERMINAL
                                (WHICH IS (= THE-TERMINAL2))))))
          (WITH VOLTAGE
                (A SUM
                   (WITH ADDEND
                         (THE VOLTAGE OF A TERMINAL
                              (WHICH IS (= THE-TERMINAL-2))))
                   (WITH AUGEND
                         (A PRODUCT
                            (WITH MULTIPLICAND (= THE-RESISTANCE))
                            (WITH MULTIPLIER
                                  (= THE-CURRENT-OF-TERMINAL1)))))))))
```

We will now look at a simple-circuit which involves two resistors connected by a 2-node as illustrated in the following diagram:



In other words the terminal2 of the first resistor is connected to the terminal1 of the 2-node and the terminal2 of the 2-node is connected to the terminal1 of the second resistor. This information can be represented easily by describing the first resistor of a simple-circuit as a resistor whose terminal2 is the terminal1 of the 2-node and by describing the second resistor as a resistor whose terminal1 is the terminal2 of the 2-node.

```
(SIMPLE-CIRCUIT
 (WITH SELF)
 (WITH RESISTOR1
       (A RESISTOR
          (WITH TERMINAL2
                (THE TERMINAL1 OF A 2-NODE
                     (WHICH IS (= THE-2-NODE))))))
 (WITH RESISTOR2
       (A RESISTOR
          (WITH TERMINAL1
                (THE TERMINAL2 OF A 2-NODE
                     (WHICH IS (= THE-2-NODE))))))
 (WITH 2-NODE))
```

Here is a dialogue with the reasoner based on these frames. First we create an instantiation for a simple-circuit and call it S-C.

```
>> S-C is (a simple-circuit)
```

One thing we can do is inquire whether S-C is indeed a circuit:

```
>> Is s-c (a simple-circuit) ?
YES
S-C IS (A SIMPLE-CIRCUIT
          (WITH 2-NODE 2-NODE-1)
          (WITH RESISTOR2 RESISTOR2-1)
          (WITH RESISTOR1 RESISTOR1-1))
```

RESISTOR1-1, RESISTOR2-1, etc. are the internal names of the object-experts that have been created for the various components of the circuit. Each of these components is worked out completely as illustrated by the following queries:

```
>> Is resistor1-1 (a resistor) ?
YES
RESISTOR1-1 IS (A RESISTOR
                  (WITH RESISTANCE RESISTANCE-2)
                  (WITH TERMINAL2 TERMINAL1-1)
                  (WITH TERMINAL1 TERMINAL1-2))
>> Is Terminal1-1 (a terminal) ?
YES
TERMINAL1-1 IS (A TERMINAL
                  (WITH VOLTAGE VOLTAGE-1)
                  (WITH CURRENT CURRENT-1))
```

etc;

Now we start supplying values to certain parts of the circuit. First we specify the voltage of the first terminal of the first resistor in the circuit:

```
>> (the terminal1 of a simple-circuit
         (which is S-C))
   is
   (a resistor
      (with terminal1
            (a terminal (with voltage 10.0))))
```

Next we specify some of the parts of the second resistor in the circuit:

INTRODUCTION

```
>> (the resistor2 of a simple-circuit
        (which is s-c))
   is
    (a resistor
          (with resistance 9.0)
          (with terminal1 (a terminal (with voltage 3.0)))
          (with terminal2 (a terminal (with voltage 0.0))))
```

At this point sufficient information is available to deduce the values of all the components of the circuit. Let us probe for one

```
>> What is (the resistance of a resistor
                (which is (the resistor1 of a simple-circuit
                              (which is s-c))) ?
21.0
```

Further justifications will show how this value was obtained.

```
>> Why ?
RESISTANCE-2 IS 21.0
BECAUSE
1.   : CURRENT-4 IS 0.333333332
2.   : AUGEND-1 IS 7.0
3.   : RESISTANCE-2 IS (QUOTIENT AUGEND-1 CURRENT-4)
>> Why ?
CURRENT-4 IS 0.333333332
BECAUSE
1.   : CURRENT-2 IS -0.333333332
2.   : SUM-2 IS 0.0
3.   : CURRENT-4 IS (DIFFERENCE SUM-2 CURRENT-2)
>> Why ?
CURRENT-2 IS -0.333333332
BECAUSE
1.   : CURRENT-1 IS 0.333333332
2.   : SUM-1 IS 0.0
3.   : CURRENT-2 IS (DIFFERENCE SUM-1 CURRENT-1)
>> Why ?
CURRENT-1 IS 0.333333332
BECAUSE
1.   : RESISTANCE-1 IS 9.0
2.   : AUGEND-2 IS 3.0
3.   : CURRENT-1 IS (QUOTIENT AUGEND-2 RESISTANCE-1)
>> Why ?
RESISTANCE-1 IS 9.0
BECAUSE
RESISTANCE-1 IS
(THE RESISTANCE OF A RESISTOR
     (WITH TERMINAL1 TERMINAL1-1)
     (WITH TERMINAL2 TERMINAL2-1))
>> Why ?
RESISTANCE-1 IS (THE RESISTANCE OF A RESISTOR
                    (WITH TERMINAL1 TERMINAL1-1)
                    (WITH TERMINAL2 TERMINAL2-1))
BECAUSE
RESISTANCE-1 IS (A RESISTOR
                   (WITH RESISTANCE 9.0)
                   (WITH TERMINAL1 (A TERMINAL (WITH VOLTAGE 3.0)))
                   (WITH TERMINAL2 (A TERMINAL (WITH VOLTAGE 0.0))))
```

INTRODUCTION

```
>> Why ?
RESISTANCE-1 IS (A RESISTOR
                    (WITH RESISTANCE 9.0)
                    (WITH TERMINAL1 (A TERMINAL (WITH VOLTAGE 3.0)))
                    (WITH TERMINAL2 (A TERMINAL (WITH VOLTAGE 0.0))))
BECAUSE YOU TOLD ME SO
```

There are many other features of the reasoning system that will be discussed later on. We hope however that these illustrations give a good initial idea on its capacity. Let us therefore start documenting the principles and the conceptual framework on which it is based.

FRAMES AND DESCRIPTIONS

2. FRAMES AND DESCRIPTIONS

It is now widely accepted that reasoning involves the construction of models based on knowledge about the domain of the model. These models and therefore also the knowledge on which they are based take the form of symbolic descriptions.

It follows that there are three types of problems we have to address:

1. Problems concerning the domain knowledge consulted to construct the model:
   - What language is used to represent knowledge?
   - How is the knowledge structured and organized?
   - How is information extracted from a model for use in later tasks?
   ...
2. Problems concerning the models themselves:
   - What is the structure of a model?
   - What are its symbolic conventions?
   ...
3. Problems concerning the process of model construction:
   - What are the principles of the construction process?
   - What kind of mechanisms interact with the model to resolve a particular task?
   ...

This chapter will work out some of the details of the first of these issues. In particular we will reflect on how knowledge needs to be represented and we will develop a concrete descriptive apparatus in the form of a knowledge representation language. The next chapter will be devoted to the description-manipulation processes: we will develop a computational architecture that is necessary and sufficient to allow the construction of reasoning systems. Subsequent chapters will make use of these two developments in order to construct and study actual reasoning systems.

This chapter is divided into two parts. In the first part, we perform a functional analysis of certain aspects of reasoning in order to determine constraints on knowledge representation. These constraints will be expressed in a series of principles.
Then we will build up a framework of concepts of knowledge representation reflecting these constraints. This framework is illustrated with concrete examples.

1. THE PRINCIPLES

1. 1. THE ISSUE OF MODULARITY

The first problem we will look at is this: It is intuitively clear for everyone who reflects on the matter that a mind has to have access to thousands of facts about all sorts of things in order to deal successfully with the problems posed by the environment in which it operates.

Here are some figures to get an idea of the magnitude of information involved. The average speaker of English uses actively about 15,000 words and understands about

25,000 words, although James Joyce used about 250,000 different words ! (Figures are taken from Ogden,1968). Each of these words has about five different meanings (actually a low estimate) which gives us 25,000 x 5 = 125,000 different shades of meaning. Each of these meanings plays a role in knowledge structures which are presumably even more extensive and are interlinked in ways that allow planning, language understanding, theory making, etc. It is clear that we are dealing here with tremendous amounts of information.

But now notice that this complexity does not bother a human mind. Only the relevant facts seem to 'spring up' when we think about a certain problem. We resolve word sense ambiguities without any apparent effort, we apply far-fetched common sense facts in order to understand even simple stories, etc.
How do we explain this efficient use of such an enormous store of facts?

A plausible explanation for this phenomenon is that the knowledge base is modularized. A particular piece of knowledge would only be accessible when the module in which it resides is active, or, as soon as a particular module becomes active, the knowledge contained in it wants to be applied. This would also explain that we sometimes have difficulty finding a fact when it is 'out of context'.

Hence the following principle:

PRINCIPLE 1:
KNOWLEDGE RELEVANT TO A CERTAIN DOMAIN IS GROUPED TOGETHER.

What exactly is meant by a domain? There seem to be two opposing views here: the *micro-world* view vs. the *general-expert* view.

> The MICRO-WORLD VIEW says that we should put every aspect of a particular class of tasks (such as playing with blocks, dealing with electronic circuits or going to a restaurant) together, i.e. knowledge about time, space, physical objects, planning capacities, etc., are all specialized for each task and grouped together in a single structure which is called a micro-world.
>
> The GENERAL-EXPERT VIEW says that we should try to construct experts for each of the aspects of reality: an expert for time, one for space, one for physical objects, etc. Faced with a particular class of tasks, the experts somehow manage to get together and solve the problems of that type of tasks.

The main advantage of the MICRO-WORLD view is that because of specialization, it is easier to tune processes and representations to the task at hand. This will give us a firmer grip on the two major problems we are facing: see to it that the model comes within physically manageable boundaries and have fast access to relevant information. The advantage of the GENERAL-EXPERT view is that because of the general applicability of the experts, the system can cope more easily with a larger number of tasks, although maybe not that well with each task. Clearly what we need is some sort of synthesis of the two views.

Let us step back here and extract a dilemma which could be called the general-special dilemma:

SPECIALIZATION leads to better performance on specific tasks but to a more restricted scope of application
GENERALIZATION leads to a wider scope of application but often to a bad performance on specific tasks.

So if you postulate a solution that is too general, you will not be able to explain how certain specific goals are realized; however if you postulate a solution that is too specialized, you cannot explain the broad scope of goals that can be realized in principle. This dilemma will occur again and again in many of the issues we will face. Its resolution is probably one of the fundamental questions of AI research. Such a resolution consists often in finding a good middle way between the two tracks set out by a particular instance of the dilemma.

One could also argue that each side of the dilemma represents different stages of development: Presumably specialized mechanisms would have to be postulated in early stages, whereas in later stages abstractions are performed leading to mechanisms with a more general scope of application. That explains on developmental grounds the necessity for constructing a synthesis of the two approaches.

1. 2. THE ISSUE OF MODULARITY CONTINUED

Grouping knowledge in terms of domains is an important step. But within each of those domains, there will still be thousands of facts, so that the complexity issue still remains unresolved. Again let us look for ways to modularize and again let us adopt the approach that the unit of organization should be related to the content of the knowledge for the same arguments given earlier in support of the previous principle:

PRINCIPLE 2:

KNOWLEDGE OF A SINGLE DOMAIN IS ORGANIZED IN UNITS THAT REFLECT THE CONCEPTUAL STRUCTURE OF THAT DOMAIN.

What will be the criterion for being a unit? There seem to be two possible solutions again: the *prototype* view vs. the *predicate* view.

The PROTOTYPE VIEW says that because we are concerned with the process of building up models of problem situations, it is natural to let the units represent prototypical model situations. In other words a unit would be a fairly extensive knowledge structure with slots for the entities that fill prototypical roles in the situation and contain ways to specify what the important questions are that have to be asked about a particular object. They would also contain information about the things which are always true for those kinds of situations, ways to specify where specialization occurs, etc.
The PREDICATE VIEW says that the main unit of organization is an atomic predicate or concept, and that all necessary and sufficient properties of this predicate should be grouped together in a single structure. Larger wholes (such as the ones laid down in

prototypes) are actively formed at the time of model construction.

Prototypes are useful because they enable the reasoner to construct whole chunks of a model at once. This is helpful for two reasons:

(i) Certain facts become immediately accessible - even if they are not yet available as premises to the model constructor. Thus we are able to solve the technical problems associated with gestalt-type reasoning.

(ii) We gain enormously in efficiency because the models are 'pre-compiled' so to speak.

Also the activation and consultation of knowledge can be made context-dependent, just as the micro-world puts constraints on what facts become available in terms of domains, the prototype puts constraints on what facts become available once inside a domain.

On the other hand the prototype approach makes it necessary to have already a great amount of ready-made models available. So there is a problem in situations where you do not have such models. This *is* a problem because obviously you cannot have encountered all possible situations you will ever be facing.
Another problem with prototypes is that in some problem situations, you may not want to invoke extensive structures because a shallow representation is sufficient. For example, in reading a sentence like

The boy-scouts on a trip to Barbados found a radio playing tunes from the Second World War in a museum belonging to a retired French captain.

one does not want to invoke possibly huge piles of information attached to boy-souts, trip-making, Barbados, radios, the Second World War, museums, captains, France...

A synthesis can be reached by adopting the prototype-principle as major organization without abandoning the possibility of postulating units for simple concepts.

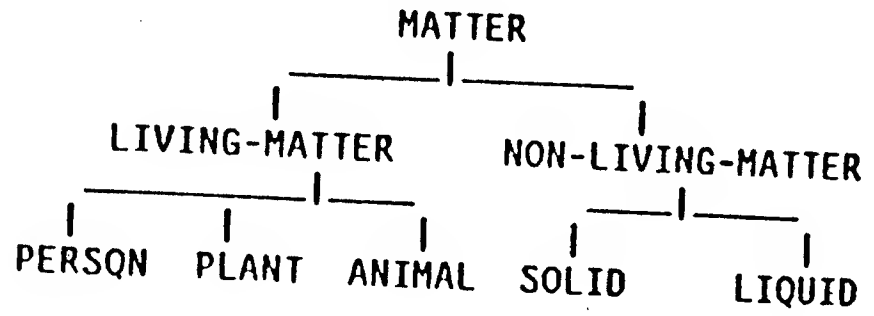
1. 3. THE ISSUE OF ORGANIZATION

So far we agreed that knowledge is grouped in domains and that knowledge inside a domain is grouped in units reflecting the conceptual structure of the domain. The question is whether this is sufficient to cope with the complexity issue, and whether there are no further principles that would contribute to a better grip on this important problem. It turns out that there is indeed a further step we can take. If the units in a domain would show some sort of internal organization, and if it would be possible to make use of this organization then we would improve our chances to deal with great amounts of information.

A study of several domains has shown that the prototypes or concepts in a domain can often be organized into *hierarchies*. The word hierarchy is ambiguous. It can refer to generalization hierarchies or to 'aspect' hierarchies. We mean hierarchy here in both senses.

A *generalization* or class/superclass hierarchy is a structuring of prototypes or concepts,

where items higher up in the hierarchy are more general and items lower down are specializations of those above. Here is a tiny example of such a hierarchy



Generalization hierarchies are important because they allow us to exploit the generalizations that are present in the domain: If certain properties are common to all concepts descending from a certain concept, then we have to store these common properties only once. For example, all properties that living things have in common would be grouped within the unit for living-matter. All depending units inherit all the descriptions attached to the units above them in the hierarchy. The unit for PERSON would automatically obtain all properties of the concept of living-matter - which automatically obtains all properties of the concept of matter, etc.

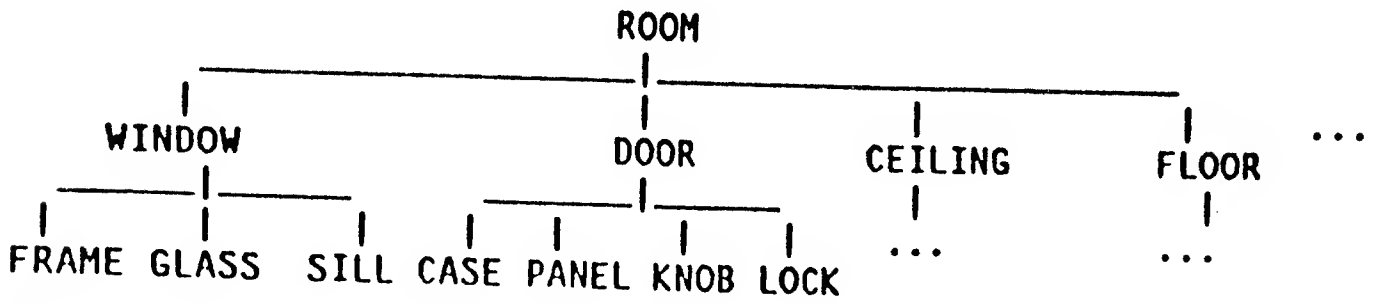
Another reason to have such generalization hierarchies is that they reflect the process of concept formation. One important type of concept formation consists in specializing a given concept or in generalizing a series of concepts based on a common property. If we organize the knowledge inside a domain in terms of structures which reflect the way they are formed, we will be in good shape to tackle these problems later.

The final reason for having generalization hierarchies is that they allow us to talk about a situation at a certain level of detail and to perform reasoning by progressively refining the answer to a given question. We will come back to this point later on.

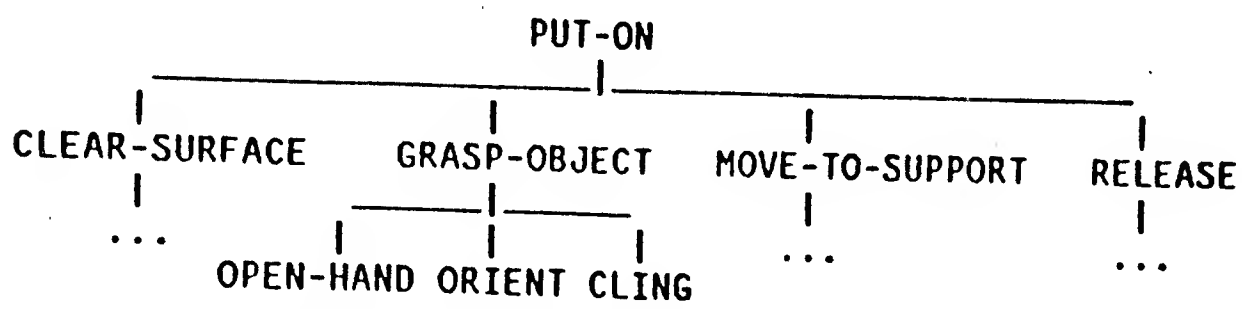
An *aspect hierarchy* is a structuring of prototypes or concepts based on the roles they play in each other. In general, an aspect is an important question that should be resolved when a particular frame is instantiated. For example, if we are reasoning about a PUT-ON action (as in planning a certain manipulation of children's blocks) these questions could be

who is the actor?
what is the object?
what is the new support?
what is the action that needs to be taken?
etc.

A typical example of an aspect hierarchy is one based on part-whole relations, like



But the word aspect indicates that 'aspect hierarchies' can be more than just part-whole or constituent structures. Height or weight of a person could be aspects that will form part of the aspect hierarchy descending from person. Or the sub-actions of a certain action are aspects working out the action itself, like in the following hierarchy:



Aspect hierarchies are important because we want the prototype structures working out aspects of a certain prototype or concept to be closely connected, so that when we are exploring a prototype, the prototypes working out its aspects will also become active. Alternatively, prototypes for parts will activate the prototype for the whole. Because aspects represent the finite set of important questions we want to ask about a certain topic we are thinking about, they organize the model in a way that helps to control the reasoning process.

These considerations lead us to adopt the following strong principle about relationships between units.

PRINCIPLE 3:
UNITS IN A DOMAIN SHOW HIERARCHICAL ORGANIZATION.

1. 4. THE ISSUE OF REPRESENTATION

We have arrived at the idea that knowledge is structured in hierarchically organized units which each contain information about a prototype or concept of the domain. The next question is how to represent this information.

Earlier on it was stated that a unit is actually a set of important questions about a particular subject-matter of the domain. An answer to such a question will be a reference to an individual of the domain of discourse. For example, the answer to the question "Who is the actor?" in a particular application of the concept PUT-ON is an individual who is the actor of that PUT-ON action. The answer to the question "What is the object?" is a reference to a particular block, the answer to "What is the old-support?" is a reference to the surface on which the block is located before the

PUT-ON operation takes place, etc.

But now we observe an interesting thing: There is usually no way to uniquely characterize the individuals of the domain. Sure, there might be a unique name, like "John_Doe", but usually objects do not have such unique names. The bed in my bedroom has no unique name. The only way I can talk about it is by indicating its role in an instance of a concept, i.e. by saying that it is the bed of my bedroom.

This is summarized in the following principle:

PRINCIPLE 5

THE INFORMATION IN A UNIT CONSISTS OF DESCRIPTIONS, WHERE A DESCRIPTION SPECIFIES THE ROLE OF AN OBJECT IN AN INSTANCE OF A CONCEPTUAL UNIT.

1. 5. MULTIPLE VIEWPOINTS

For most nontrivial problem areas, a single viewpoint on the objects is insufficient to effectively describe the object, because it usually plays a role in many different prototypes, where each prototype implies a particular way of conceptualizing a certain piece of reality.

For example, we may think of a natural language sentence as describable from the viewpoints of phonetics, phonology, morphology, constituent structure, grammatical categories (like noun or verb), grammatical relations (like subject or predicate), case relations (like agent or patient), organizational structure (like theme/rheme or topic/focus), etc. All these different viewpoints determine a specific aspect of the linguistic forms occurring in the sentence and each of the viewpoints refers to a particular aspect of its meaning.

The curious thing is that often it is impossible to do analysis or synthesis of an object without maintaining models that describe the object from all its different viewpoints. This is so because one viewpoint does not contain enough information to proceed and only a cooperation between the different viewpoints provides enough constraints to come to a solution.
For example one soundstructure may lead to many possible phonological structures, a phonological structure may lead to many different morphemes, one morpheme may have a variety of different meanings, a certain constituent structure may embody several different dependency structures, a declarative sentence structure may signal a variety of speech acts, etc.
Each of these structures can be constrained by structures from other viewpoints.
For example, a declarative sentence structure can in principle signal a question but additional constraints from intonation may help to make a decision anyway.

But in order to explore these inter-viewpoint relationships we have to maintain a description from multiple viewpoints, hence the following principle:

PRINCIPLE 6:

A MODEL OF A PROBLEM-SITUATION CONTAINS A DESCRIPTION OF THE OBJECTS FROM VARIOUS VIEWPOINTS.

It is important to realize that the need to maintain a description from multiple viewpoints has been recognized in all micro-worlds that have been studied so far, and it is probably an epistemological universal.
In reasoning about circuits we want to maintain descriptions from the viewpoints of DC analysis, HF analysis, midband analysis, S-plane analysis, etc.
In geometry, we want to describe and reason about objects in Euclidian terms (using concepts like LINE, ANGLE, etc.) and Cartesian terms (using X-Y coordinates).
And similarly for other domains.

DISCUSSION

The alternative to modularization is to have a 'flat' representation, such as a database in the form of a linear list of patterns, a set of nodes and links without further internal structure, and rules in the form of linear lists without any further organization. Examples of non-modularized/non-organized systems are resolution theorem provers (Robinson,1965), early pattern-directed invocation systems (such as Hewitt,1969, et.al.), the first generation of production systems (Newell and Simon,1972) and early semantic networks (such as Quillian,1968).

All these researchers were well aware of the complexity problem but they assumed that it was a technical problem which could be overcome by finding suitable mechanisms. Examples of such mechanisms are hash-coding of the database of patterns (as discussed in McDermott,1975), powerful computational mechanisms like parallel marker passing in semantic networks (as discussed in Fahlman,1977), a.o.

This view is not shared by another group of people who believe that an important principle must be behind our dealing with complexity. This principle is modularization and organization of knowledge. The modularization movement was lead by Minsky's frame-paper (Minsky,1974). Right now there seems to be widespread agreement that modularization and organization is necessary: production systems are organized in packets (Lenat,1976), semantic networks are partitioned in spaces (Hendrix, 1975), etc.

The micro-world view is developed in Minsky and Papert (1971). and is vividly illustrated in Winograd (1972). A related notion is that of a problem-space (Newell and Simon,1972). The general expert view is for example advocated and illustrated in Kahn (1975) who has constructed a time-specialist.

Arguments between prototype and predicate advocates were the focus of attention in the mid seventies. The major first paper on prototypes was Minsky's influential frame-paper (Minsky,1974). The predicate view had before been advocated by systems based on predicate calculus or organized around 'conceptual primitives' like Schank [1975] and Wilks [1972]. See Wilks (1976) for arguments against the prototype-hypothesis. At present, the matter seems to have been resolved in favour of the prototype view. For example, researchers who concentrated on conceptual

primitives earlier on have now incorporated these primitives in a broader framework that uses prototype-like structures such as plans and scripts (Schank and Abelson, 1977) or pseudo-texts (Wilks,1978). Note also that the notion of a frame is very similar - if not identical - to Piaget's concept of a schema (see e.g. Piaget, 1968). The importance for problem solving of finite models of the domain, i.e. the principle that we should organize knowledge such that a finite number of questions needs to be asked is a central hypothesis in Minsky's frame-theory (Minsky,1974).

The idea of a hierarchy is a very old one and has been an important part of research on semantic networks (cf. Quillian (1968)) or frames (Minsky,1974). The idea also developed within the context on research on problem solving itself. In fact the difference tables of the means-ends analysis of GPS (Ernst and Newell,1969) leads naturally to the notion of a hierarchy of actions that are discriminated by the purpose of the action. Such a structure is further worked out by Rieger (1975) who talks about bypassable causal selection networks, stressing the fact that although the hierarchy is the main skeleton for organization, there must be ways to bypass that structure.

The principle of hierarchy has also been realized by other disciplines which have to cope with large amounts of information, such as catalogues in libraries, language thesauri like Roget's, corpora of legal laws, etc. It also interesting to observe that Simon (1972) introduces modularization and hierarchy as the two prime methods for reducing complexity in any system and he links the success in evolution to these two factors.

The idea of multiple-viewpoints was introduced by Moore and Newell(1973) and formed an important part of the heterarchy-idea (cf. Minsky and Paper (1971), Winograd(1972)). Another word for viewpoint is knowledge source (cf. Erman and Lesser (1975)). Multiple viewpoints are the focus of attention in recent knowledge representation efforts, especially KRL (Bobrow and Winograd,1977).

2. THE FRAMEWORK

We discussed a number of issues in the previous paragraphs and argued for a particular stand on each issue. Each stand was then expressed as a certain principle. We will now develop a number of theoretical concepts which substantiate the principles advanced so far.

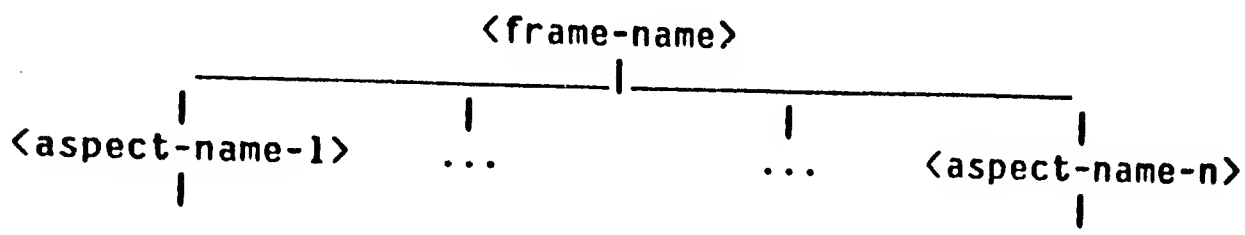
2. 1. FRAMES

The first concept, that of a *frame*, is postulated in the light of the principle that knowledge of a domain is organized in units reflecting the conceptual structure of the domain.

A frame is a structure that contains information about a particular conceptual unit of the domain under study. A frame has a frame-name and a number of aspects or slots. A slot is filled by entities that play a particular role or illuminate a certain aspect of the situation described by the frame. These aspects represent the finite set of questions or the components of a certain prototype - thus realizing the principle of hierarchical organization of knowledge (hierarchy in the sense of aspect-hierarchy). We will give

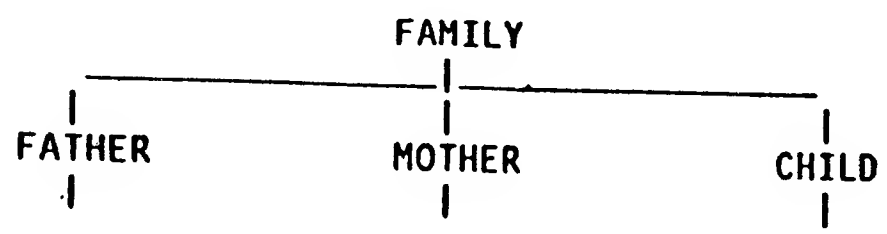
names to such aspects and call them aspect-names. To each slot information is attached about the entities that fill the slots.

A frame can be visualized as follows

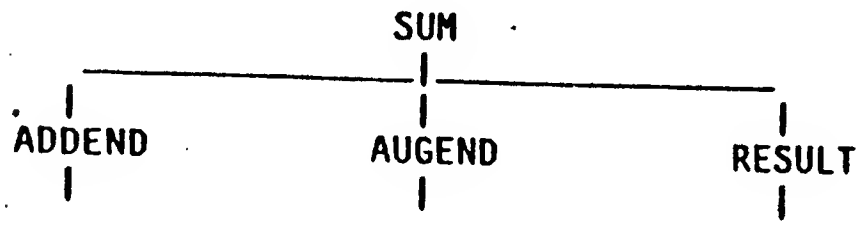


Here are some examples:

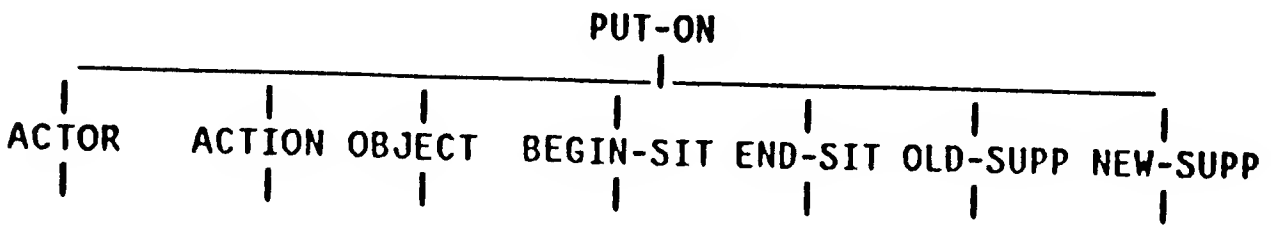
+ A frame for a prototypical family with roles for a father, mother and child:



+ A frame for the arithmethic operation SUM with aspects for the addend augend and result:



+ A frame for the action of putting an object which is resting on an old-support in a certain begin-situation on a new-support in a certain end-situation:



As we go on we will develop a language that enables us to represent information along the ways prescribed by the theory. So for each theoretical concept, we will introduce a syntactic notation.

The syntactic form of a frame will be a list structure

```
(<frame-name>
 (WITH <aspect-name-1>)
 ...
 (WITH <aspect-name-n>)).
```

The ordering of the aspect specifications does not affect the meaning.

For example, the syntactic representation of the FAMILY-frame is

```
(FAMILY
 (WITH FATHER)
 (WITH MOTHER)
 (WITH CHILD)).
```

Here is the PUT-ON frame:

```
(PUT-ON
      (WITH ACTOR)
      (WITH ACTION)
      (WITH OBJECT)
      (WITH OLD-SUPPORT)
      (WITH NEW-SUPPORT)
      (WITH BEGIN-SITUATION)
      (WITH END-SITUATION)).
```

What is the meaning of these representations? As said before, intuitively speaking a frame represents a class of typical situations. If we would want to explicate in terms of a Tarskian-style model theoretic semantics what is meant by this, we would say that the extension of a frame is a set of configurations of individuals which are in the situation indicated by the concept of the frame.

Let us call such a set of configurations the instantiation-set of the frame. Each configuration will be called a possible instantiation. The set of individuals that may fill up a particular position (i.e. may fill a particular slot in a frame) are called the set of possible slot-fillers of that slot.

For example, the instantiation-set of the SUM-frame is the set of numbers which are in the sum relation: {<2,1,1>,<3,1,2>,<3,2,1>,...}. A particular configuration like <2,1,1> is a possible instantiation of SUM. The set of numbers that can be the addend of a sum: {1,2,...} are the possible slot-fillers of the ADDEND slot.

THE SELF-SLOT

Often it is necessary to talk about an instance as such - without specializing to a particular role. For example we might want to attach properties to the room-frame that every room will have, or we might want to refer to an object as a room, not the window of a room or the door of a room, but simply the room itself. There are two ways to incorporate these capabilities. Either we can attach descriptions to the room-frame as such and introduce special mechanisms to talk about instances, or we can introduce a special aspect, which will further be called the self-aspect or self-slot, which has as its filler an object that is an instance of the concept mentioned in the frame.

The advantage of this second method is that we gain in economy of representation (which

will lead to a simpler activation mechanism) without losing in expressability. That is why we will adopt this method. So every frame will have a self-aspect, as in

```
(FAMILY
  (WITH SELF)
  (WITH FATHER)
  (WITH MOTHER)
  (WITH CHILD)).
```

The self-slot in the family frame is filled by objects which are a family.

We now develop a descriptive apparatus to refer to individuals based on the role they play in an instance of a concept and we will show how such descriptions can be used to constrain what can be the answer to a particular question in a frame.

2. 2. BASIC DESCRIPTIONS

A *description* is a construct used to refer to an individual by saying that it plays a particular role, i.e. fills a certain aspect in an instantiation of a frame. This role will be called the view.

The syntactic representation of a description is constructed by taking the syntactic representation of a frame and putting the name of the aspect that is the view in the description in front:

```
(<first-article> <VIEW> OF <second-article> <FRAME-NAME>
        (WITH <ASPECT-NAME-1>) .... )
```

as in

```
(THE MOTHER OF A FAMILY
     (WITH FATHER)
     (WITH CHILD)).
```

The articles give a hint on how the referent of the description should be found. If the first article is a definite article (i.e. THE), then the description is a definite description, i.e. the referent is a specific individual. If the first article is an indefinite article (i.e. A or AN), then the description is an indefinite description, i.e. the referent is an arbitrary individual from the range of the description.
For example, because a family can have many children, we might have a description like

```
(A CHILD OF A FAMILY
   (WITH FATHER)(WITH MOTHER))
```

The second article (in front of the frame-name) gives a hint on the status of the instance as a whole. If this instance is definite we use the definite article, if it is not we use the indefinite article. For example, if the family can be sufficiently constrained to be a particular one, we might say something like

```
(A CHILD OF THE FAMILY ...)
```

If the concept used in the frame is an individual-concept, i.e. there is only one object that can fill its self-slot, then we write no article at all. (More about individual-concepts will be said later on.)
For example suppose we have a frame for JOHN:

```
(JOHN
  (WITH SELF)
  (WITH AGE))
```

Then we can refer to the age of John by saying

```
(THE AGE OF JOHN)
```

A *partial description* is a description where not all aspects that are in the frame of the concept used in the description are mentioned. For example, we could refer to

```
(THE DOOR OF A ROOM)
```

without saying anything about the window, floor, ceiling, etc.

A second type of basic descriptions allows us to refer to a particular instance itself. Such a description is written as

```
(<article> <FRAME-NAME>
     (WITH <ASPECT-NAME-1>) ...)
```

Where the article is the definite article if the referent is definite, an indefinite article if the referent is an arbitrary element of the range of the concept and no article if the concept is an individual concept.

The following are examples of basic descriptions referring to an instance of a concept:

```
JOHN
(A ROOM)
(A FAMILY)
```

Note that a description has the flavor of a function. Indeed if the description is guaranteed to point uniquely to a particular individual, it *is* a function that has this individual as value. However in most cases a description does NOT characterize a unique individual. The only thing we can say in such cases is that it characterizes a typical member of the set of individuals for which the description holds.

Descriptions can be used to further constrain the range of a description by being attached to a slot in a description. Attachment will be represented by writing the description after the aspect-name, as in

```
(A FAMILY (WITH FATHER JOHN) (WITH MOTHER MARY))
```

which can be read as "a family whose father is John and whose mother is Mary". The descriptions attached to the father and mother aspect respectively refer to an object which is an instance of John and an instance of Mary.

When a description is attached to an aspect in a frame, it means that all possible slot-fillers of that aspect are described by that description. In this case attachment works like implication. For example if we want to say that every mother of a family is a female-person, where there is a frame for female-person that looks like this:

```
(FEMALE-PERSON
    (WITH SELF))
```

then we can do so by attaching the description (A FEMALE-PERSON) to the mother slot of the family-frame:

```
(FAMILY
  (WITH SELF)
  (WITH MOTHER
        (A FEMALE-PERSON))
  (WITH FATHER))
```

It is also possible to further constrain the instance itself by writing

(<relative-pronoun> IS <description>)

after the frame-name in the description. The relative-pronoun is WHO if the instance refers to a person, otherwise it is WHICH.
Suppose, for example, that we had a frame for MOTHER:

```
(MOTHER
  (WITH SELF)
  (WITH CHILD))
```

then we can refer to an object by saying

```
(A CHILD OF A MOTHER
   (WHO IS MARY))
```

Before we introduce other types of descriptions it must be remarked that the use of articles, pronouns and the copula IS is syntactic sugar that is completely ignored by the reasoner. For example, a description of the form

```
(THE CHILD OF A FAMILY)
```

is internally represented as

```
(CHILD FAMILY)
```

Or a description of the form

```
(A CHILD OF A MOTHER (WHO IS MARY))
```

is represented as

```
(CHILD MOTHER (WITH SELF (SELF MARY)))
```

The more elaborate syntax has been introduced for the convenience of the reader and the user of the reasoning system.

2. 3. ASPECT-SPECIFICATIONS

PROJECTIVITY

Now we observe that it is usually possible to divide the set of instantiations of a frame into groups, further called instantiation-groups. Each instantiation in such a group has a certain set of aspects in common.

Here is an example. Consider a frame for family with aspects for the mother, the father and a child. Because every family has only one mother and one father, but possibly many children, it makes sense to divide the instantiation-set of the family frame into groups where each group corresponds to one family. All instantiations in this group have the same father and the same mother. They differ in that the child-slot could be filled by a different individual. We will call the aspects that have the same filler in a given instantiation-group *projective* aspects. The other aspects are called non-projective. It is

important for the reasoner to know this because as soon as it knows one instance of an instance-group it can infer the slot-fillers of the projective aspects of all members of that group. It turns out that this knowledge is crucial for two reasons: (i) It is an important aid in determining whether a description is definite and in finding consequently the referent of the description, and (ii) it is one of the tools with which two objects can be shown to be identical.

For example, if a certain person is described as

```
(THE MOTHER OF A FAMILY (WITH FATHER JOHN))
```

and a little while later this same person is described as

```
(THE MOTHER OF A FAMILY (WITH FATHER MR-JONES))
```

Then we know that John and Mr-Jones are referring to the same person because the father aspect of a family is projective. This means that either if Mr-Jones was already known to be a certain object and John was already known to be a certain different object, then from now these objects should be considered identical. We say in such a case that the two objects are *merged*. Or if Mr-Jones was not yet known, then the description Mr-Jones should be predicated for John or vice-versa.

CRITERIALITY

But how does the reasoner know under what circumstances two instances belong to the same group? In other words, how do we specify what the criterion for dividing the set of instantiations into groups is? Another property of aspects takes care of this. Often a certain aspect can only once be filled by a certain individual. Somebody can only be the mother in one family for example. We say in such a case that this aspect is *criterial*.

But sometimes more than one aspect has to co-operate in order to find what instantiation-group is intended. In a frame of LINE-SEGMENT like

```
(LINE-SEGMENT
   (WITH SELF)
   (WITH BEGIN)
   (WITH END)
   (WITH DISTANCE)),
```

begin and end are criterial because there are no two line-segments with the same begin and the same end. In the same frame the self-aspect is also criterial. But the begin aspect on its own is not criterial, because there can be two lines with the same begin.

The importance of knowing what series of aspects are criterial is that they provide essential information for finding the instantiation(group) of a description.
Take for example the following description:

```
(A LINE-SEGMENT
   (WITH BEGIN POINT1)
   (WITH END POINT2))
```

In order to find out what line-segment is referred to, the reasoner looks at a series of aspects that are criterial and for which the description contains specific information. In the example given here the reasoner knows that begin and end are criterial, it can therefore look whether the objects point1 and point2 are described already as the

begin and end of a line-segment. If that is so it knows already about this particular instance of the line-segment concept and can infer the other individuals in this instantiation.

Criteriality is also the prime mechanism to know whether two descriptions have to be *merged* or not. Two descriptions are merged if they refer to the same instance. For example, when a particular object is at one point described as

```
(A LINE-SEGMENT
   (WITH BEGIN POINT1)
   (WITH END POINT2))
```

and at another point as

```
(A LINE-SEGMENT
   (WITH DISTANCE 3_CM))
```

then we know that each time we are talking about the same line-segment. We know this because the SELF-slot of line-segment is criterial and therefore the object can only once be described as a line-segment.
In contrast,

```
(THE FATHER OF A FATHER-CHILD-RELATION
     (WITH CHILD GEORGE))
```

and

```
(THE FATHER OF A FATHER-CHILD-RELATION
     (WITH CHILD JOHN))
```

do not have to merged because somebody can be the father of more than one father-child-relation, i.e. the father slot in the frame for father-child-relation is non-criterial.

It is interesting to observe that although in natural languages articles give a hint on the status of a description, information of what aspects are criterial is a much more robust way to figure out whether a description is supposed to be definite or not. That is why we are able to ignore the articles that are given in a description. That may also be the reason why certain languages (such as classical Latin) do not have articles but can still function properly.

INDIVIDUALITY

A strong form of criteriality is when it is not only the case that an individual can only once fill a certain aspect in an instance but that there is only one individual in the domain that can ever fill this aspect. In such a case we say that this aspect is *individuating*. We will call a concept whose self-aspect is individuating an *individual-concept*.

The importance of knowing whether an aspect is individuating is that it tells the reasoner which descriptions are individual-descriptions, i.e. which descriptions refer to a unique individual. We will see later that the reasoner builds up a list of individuals which are accessible by way of these individual-descriptions, so that the referent can be retrieved when the description occurs.

A further refinement of the notion of an individuating aspect is to specify that it is a

unique description with respect to a certain group of individuals. For example, 1 is an individual concept referring to a particular number. And so is 2. In order to determine by matching that 1 and 2 are different, we would need to attach a description to 1 saying (NOT 2), and similary for all other numbers. However if we say that 1 is an individual description with respect to the numbers, and 2 is another individual description with respect to the numbers, then we can construct the matcher in such a way that it will consider every individual introduced by an individual concept which is unique with respect to a certain group as different from any other individual introduced by an individual concept of that group.

CONVENTIONS

Let us now establish some conventions for representing this information. What we will do is add a list of so-called aspect-specifications at the end of an aspect-list. The aspect-specifications are a list of specifications of the form (<type-of-specification> <case-1> ... <case-n>) where a case is an aspect or a list of aspects if the type-of-specification is criteriality, that satisfy the property mentioned in the <type-of-specification>.
The type-of-specification is
CRITERIAL if the cases indicate series of aspects which are criterial.
NON-PROJECTIVE if the cases are non-projective aspects.
INDIVIDUATING if the cases are individuating aspects.

For the LINE-SEGMENT frame this leads us to

```
(LINE-SEGMENT
    (WITH SELF)
    (WITH BEGIN)
    (WITH END)
    (WITH DISTANCE)
    (ASPECT-SPECIFICATIONS:
       (CRITERIAL: (BEGIN END) (SELF))))
```

The default for projectivity is projective. The default for individuality is non-individuating and the default for criteriality is non-criterial. If the default is not violated we do not write any aspect-specifications. In the rest of the text these aspect-specifications will only be written if they are needed to prove a point.

Here are some more examples: A frame for the individual concept John_Doe:

```
(JOHN_DOE
  (WITH SELF)
  (ASPECT-SPECIFICATIONS:
    (INDIVIDUATING: SELF)))
```

When we want to add that the self-slot of JOHN_DOE is individuating with respect to people, we write this as follows:

```
(JOHN_DOE
  (WITH SELF)
  (ASPECT-SPECIFICATIONS:
    (INDIVIDUATING: (WITH-RESPECT-TO PEOPLE SELF))))
```

An individual concept can be individuating with respect to a variety of domains.

The family frame with projectivity and criteriality declarations:

```
(FAMILY
   (WITH SELF)
   (WITH FATHER)
   (WITH MOTHER)
   (WITH CHILD)
   (ASPECT-SPECIFICATIONS:
    (CRITERIAL: (FATHER)(MOTHER) (CHILD)(SELF))
    (NON-PROJECTIVE: CHILD)))
```

2. 4. THE CONNECTIVES

The next extension of the descriptive apparatus will enable us to specify tighter constraints on a particular slot-filler by the use of more than one description which is combined with one of the logical connectives AND, OR, XOR or NOT.

A description like

```
(AND (THE FATHER OF A FAMILY
            (WITH MOTHER MARY))
     (THE CHILD OF A FAMILY
            (WITH FATHER JOHN))).
```

refers to an object as being the father of a family whose mother is Mary AND the child of a family whose father is John. In other words the interpretation of a conjunction of descriptions is an individual out of the intersection of the possible slot-fillers of the view of each description.

The disjunction of descriptions, represented with the connective OR, is an individual out of the union of the possible slot-fillers of the view of each description.

Finally, an exclusive-disjunction of descriptions, represented as XOR, is an individual out of the range of the first description, or the range of the second description but not of the intersection of these two ranges.
For example, a person can be described as either being a male or a female person:

```
(PERSON
 (WITH SELF
       (XOR (A FEMALE-PERSON)
            (A MALE-PERSON))))
```

Sometimes we want to express the fact that an entity does not play a role in a certain instantiation of a frame. This type of specification is particularly relevant if we are working with incomplete worlds, i.e. when we have insufficient information to construct a complete closed model of a certain problem situation.

Such a negative description will be expressed by a description of the form (NOT <description>) where <description> is a description. For example,

```
(NOT (THE FATHER OF A FAMILY
           (WITH MOTHER JOAN)))
```

is a negative description. Its extension is an individual which is not the father in an

instantiation of the family-frame where an individual which has the name of Joan plays the role of the mother.

The relationships between connectives known from propositional calculus hold for descriptions with connectives.
For example, based on DeMorgan's law

```
(OR (NOT (A FEMALE-PERSON))
    (NOT (A MALE-PERSON)))
```

is equivalent to

```
(NOT (AND (A FEMALE-PERSON)
          (A MALE-PERSON)))
```

2. 5. CO-REFERENTIAL DESCRIPTIONS

The primitive semantic relation of the description system is co-referentiality which says that two descriptions refer to the same individual, or that two descriptions have the same individual as referent. The way co-referentiality was expressed so far is by attachment: When we attach a description to an aspect of a frame, we say in effect that the individual which is an answer to the aspect is co-referential with the description that is attached to the aspect.
For example in

```
(THE RESULT OF A WRITE
     (WITH ACTOR (A PERSON)))
```

The actor of this write-instance is specified as being co-referential with a filler of the self-aspect of a person-instance.

Attachment allows us to express co-referentiality between one aspect of an instance of a frame and one aspect of an instance of another frame. But in many cases we want to express co-referentiality between more than one aspect. Here is an example of a representation problem where this is needed. Suppose we have the concept of a parent-child-relation as in

```
(PARENT-CHILD-RELATION
        (WITH SELF)
        (WITH PARENT)
        (WITH CHILD))
```

and the concept of a MOTHER-CHILD-RELATION with aspects for the mother and the child as in

```
(MOTHER-CHILD-RELATION
        (WITH SELF)
        (WITH MOTHER)
        (WITH CHILD)).
```

Now we want to specify that the pair mother-child corresponds to the pair parent-child. Note that something like

```
(MOTHER-CHILD-RELATION
        (WITH SELF)
        (WITH MOTHER
              (THE PARENT OF A PARENT-CHILD-RELATION))
        (WITH CHILD
              (THE CHILD OF A PARENT-CHILD-RELATION)))
```

is not a sufficient constraint because it does not say that the mother is the parent of the same instantiation of the parent-child-relation as the child is the child of.

What we will do is say that the child slot of the parent-child-relation in the description attached to the mother slot is co-referential with the child slot in the mother-child-relation frame. Co-referential links will be represented by writing

```
(= <REFERRING-NAME>)
```

after each slot that is co-referentially related. Such an expression is called a co-referential description. The referring-name is lexically scoped within one frame. For the MOTHER-CHILD-RELATION frame, this leads us to

```
(MOTHER-CHILD-RELATION
   (WITH SELF)
   (WITH MOTHER
         (THE PARENT OF A PARENT-CHILD-RELATION
               (WITH CHILD (= THE-CHILD))))
   (WITH CHILD (= THE-CHILD))).
```

This frame now expresses the fact that the mother of every mother-child-relation is the parent of a parent-child-relation whose child is co-referential with the child of the mother. Observe the co-referential descriptions attached to the child-slot in the mother-child-relation frame and the child-slot in the description based on the parent-child-relation frame.

The default name for the fillers of the frame itself is

```
(= THE-<aspect-name>)
```

For example, the default name for the filler of the child aspect in a mother-child-relation is THE-CHILD. We therefore can write

```
(MOTHER-CHILD-RELATION
   (WITH SELF)
   (WITH MOTHER
         (THE PARENT OF A PARENT-CHILD-RELATION
               (WITH CHILD (= THE-CHILD))))
   (WITH CHILD))
```

Here is another example of co-referential links inside a frame. An uncle of a niece-or-nephew can be described as the brother of the parent of this niece-or-nephew or the husband of the aunt of this niece-or-nephew:

```
(UNCLE
  (WITH SELF
        (OR (A BROTHER
               (WITH BROTHER-OR-SISTER
                     (A PARENT (WITH CHILD
                                     (= THE-NIECE-OR-NEPHEW)))))
            (A HUSBAND
               (WITH WIFE
                     (AN AUNT (WITH NIECE-OR-NEPHEW
                                    (= THE-NIECE-OR-NEPHEW))))))
  (WITH NIECE-OR-NEPHEW (= THE-NIECE-OR-NEPHEW)))
```

Where reference was made to a frame for brother:

```
(BROTHER
   (WITH SELF)
   (WITH BROTHER-OR-SISTER))
```

husband:

```
(HUSBAND
   (WITH SELF)
   (WITH WIFE))
```

and aunt:

```
(AUNT
 (WITH SELF)
 (WITH NIECE-OR-NEPHEW))
```

There are some important issues that are underlying the representation mechanisms proposed so far. One of them is that we insist on local rather than global contexts: An aspect has only meaning within the context of the frame in which it is located. A co-referential description has only meaning (i.e. a binding) within the local text that surrounds it, etc. The major reason for doing this is because we fear that in a large knowledge system the occurrence of global elements is dangerous, especially if the system is developed over a longer period of time by several people. Because the representations are local, the writer of descriptions here only has to worry about the immediate text.

2. 6. CONDITIONAL DESCRIPTIONS

Conditional descriptions are necessary when we want to make a predication conditional on the presence of a certain constraint. For example, we might want to represent the information that a parent is a mother when she is a female-person.

Conditional descriptions are expressed as follows. First we introduce the entity for which the conditional holds by using a referring-name introduced by a co-referential description. Then we give a list of pairs where the first element is the condition that the entity has to satisfy in order for the second element to be a valuable description. The condition-indicator in front denotes the type of condition.

All this is represented as

```
(<CONDITION-INDICATOR> <REFERRING-NAME> IS
          (<CONDITION-1> <RESULTING-DESCRIPTION-1>)
          ...
          (<CONDITION-N> <RESULTING-DESCRIPTION-n>))
```

As in

```
(IF THE-PARENT IS
    ((A FEMALE-PERSON)
     (A MOTHER (WITH CHILD (= THE-CHILD)))))
```

which says that if the referent of THE-PARENT is described as a female-person, then this description is equivalent to

```
(A MOTHER (WITH CHILD (= THE-CHILD)))
```

If we now attach this description to a frame:

```
(PARENT
  (WITH SELF (= THE-PARENT)
        (IF THE-PARENT IS
            ((A FEMALE-PERSON)
             (A MOTHER (WITH CHILD (= THE-CHILD))))))
  (WITH CHILD (= THE-CHILD)))
```

then this means the same as saying that every parent who is a female-person is a mother whose child is the child of the parent.

When the condition in a conditional contains co-referential descriptions that do not yet have a specific reference, then they are 'bound' by a matching process to a particular individual and aspects which occur within the scope of that condition and which are co-referential are then known to have the same individual as referent.

r example, in

```
(PERSON
  (WITH SELF (= THE-PERSON)
        (IF THE-PERSON
            ((A FATHER
                (WITH CHILD (= A-CHILD)))
             (A PARENT
                (WITH CHILD (= A-CHILD)))))))
```

The co-referential-description (= A-CHILD) will obtain a specific reference when the condition itself is compared with a description predicated for the-person. Later references obtain the same referent. So if the following description holds for the referent of THE-PERSON:

```
(A FATHER (WITH CHILD JOHN))
```

then the resulting-description will be

```
(A PARENT (WITH CHILD JOHN))
```

Let us explore further the various components of a conditional description. First the condition-indicator. There are two dimensions along which conditional descriptions can be classified. The first dimension is based on the order in which the conditions are tried out. Either we can try out all conditions at once, in which case we have a *parallel conditional*, or we can try one condition after the other until one matches. This will be called a *sequential conditional*.

The second dimension concentrates on the time for which the conditional has to remain active. Here we see three possibilities: Either the conditional is exercised only once, i.e. at the time it is presented to the reasoner (e.g. at the moment a description is instantiated). We will call this an *instantaneous conditional.* Or it is exercised until a description is found that matches one of its conditions after which it is aborted, we will call this a *single-event conditional.* Or it is exercised continuously, i.e. even if a match is found the conditional remains under consideration by the reasoner. We will call this a *continuous conditional.*
Each of these types has different applications. For example, the first one can be used to look around in the state of a model at a certain moment of time, and make a decision based on that state. The second one is used to derive a certain description but we know that this derivation will need to take place only once. Finally the third type is useful to construct generators which collect all possible cases satisfying a certain condition, or demons which continually watch out for a certain condition to hold.

Moreover there is a natural relation between these conditionals. Clearly an instantaneous conditional is only useful if sequential. The reasoner should look around for cases until it finds one that matches and then it should stop looking. On the other hand a continuous conditional has to be parallel because each of the conditions will generate special cases. Finally the single-event conditional has to be parallel too because we want to wait until one of the conditions is true and then perform an action. So we end up with three types of conditionals: instantaneous/sequential, continuous/parallel and single-event/parallel. For each of these types we introduce an indicator:
IF-NOW for instantaneous/sequential conditional
IF for single-event/parallel conditional
WHEN for continous/parallel conditional.

Consider for example,

```
(PARENT
  (WITH SELF (= THE-PARENT)
        (IF THE-PARENT IS
            ((A FEMALE-PERSON)
             (A MOTHER (WITH CHILD (= THE-CHILD))))))
  (WITH CHILD))
```

This description says that when the individual which is the referent of the-parent is known to be described as a female-person, then the description

```
(A MOTHER (WITH CHILD (= THE-CHILD)))
```

holds for the filler of the self-slot. Clearly it is only necessary to exercise this conditional once, i.e. as soon as we know that the person is a female-person, we can deduce that she is a mother and there will be no necessity to deduce that fact again later. On the other hand we have to wait until the condition is true because it might not be known at the time the condition enters the reasoner whether the person is female or not.
Here is a conditional with two cases.

```
(PARENT (= THE-PARENT)
  (WITH SELF (= THE-PARENT)
        (IF THE-PARENT IS
            ((A FEMALE-PERSON)
              (A MOTHER (WITH CHILD (= THE-CHILD))))
            ((A MALE-PERSON
              (A FATHER (WITH CHILD (= THE-CHILD)))))))
  (WITH CHILD))
```

Now comes an example of a continuous, parallel conditional. Suppose we want to express transitivity of a certain relation in a way that causes the addition of a new description each time the parts are known. For example, suppose we are working in the blocksworld domain and we want to set up a frame such that when an object is above another object and this object is above another object itself, then the first object is above the latter object. This can be accomplished as follows:

```
(ABOVE-RELATION
  (WITH SELF)
  (WITH LOWER-OBJECT
        (WHEN THE-UPPER-OBJECT IS
              ((THE LOWER-OBJECT OF AN ABOVE-RELATION
                    (WITH UPPER-OBJECT (= ANOTHER-OBJECT)))
               (THE LOWER-OBJECT OF AN ABOVE-RELATION
                    (WITH UPPER-OBJECT (= ANOTHER-OBJECT))))))
  (WITH UPPER-OBJECT))
```

The above-relation has aspects for a lower-object and an upper-object. Attached to the lower-object aspect is a conditional description which looks out whether the upper-object is described as the lower-object of an above-relation. As soon as that is the case the above-relation is asserted between the lower-object of the original above-relation and the upper-object of this newly discovered above-relation. The point is that for every possible instance of an above-relation that will be discovered this new above-relation will be asserted.

Now we turn our attention to the conditions in a conditional-description. So far it is assumed that they are basic descriptions. We now extend the notion of a condition by allowing that it can be a combination of descriptions using the logical connectives AND, OR, XOR and NOT and that the last condition can be the special indicator ELSE which is a condition that is always true.

When the condition is a conjunction then the object referred to by the referring-name has to be described in terms of all of the conjuncts in order for the corresponding resulting-description to match. When it is a disjunction then the object has to be described in terms of any of the disjuncts, and when it is an exclusive disjunction, it has to be described in any of the disjuncts and in the negation of the others.

The following is an example of a more complex condition:

```
(PERSON (= THE-PERSON)
  (WITH SELF
        (IF THE-PERSON IS
            ((AND (A MALE-PERSON)
                  (A PARENT
                     (WITH CHILD (= THE-CHILD))))
              (A FATHER (WITH CHILD (= THE-CHILD)))))))
```

2. 7. EXPLICIT PREDICATION

It sometimes happens that the conditional descriptions become very complicated. In such cases it is often no longer clear what the object is to which the final resulting-description has to be predicated. In order to make that clear again, we introduce the notion of *explicit predication.*

An explicit predication is of the form

```
(<a-referring-name> IS <description>)
```

As in

```
(THE-PERSON IS (A FATHER (WITH CHILD (= THE-CHILD))))
```

A referring-name is a name introduced by a co-referential description somewhere else in the frame. The meaning of an explicit predication is simply that the description holds for the referent of the referring-name. In other words that the referent of the referring-name is co-referential with the referent of the description.
In this case that the description

```
(A FATHER (WITH CHILD (= THE-CHILD)))
```

holds for the object referred to by THE-PERSON.

Thus the previous frame for person could have been written as

```
(PERSON
  (WITH SELF (= THE-PERSON)
        (IF THE-PERSON IS
            ((AND (A MALE-PERSON)
                  (A PARENT
                     (WITH CHILD (= THE-CHILD))))
              (THE-PERSON IS
                    (A FATHER (WITH CHILD (= THE-CHILD))))))))
```

2. 8. HIERARCHY

Another important principle of the frame-theory which underlies the language presented in this section is that knowledge is organized in generalization or class/superclass hierarchies. The idea is that frames lower in the hierarchy inherit descriptions attached to generalizations of this frame higher up in the hierarchy. Fortunately the descriptive apparatus developed so far contains the tools needed to express hierarchies.

The way we represent hierarchies is by describing a particular aspect of a frame in terms of another frame, i.e. by attaching a description to that aspect and by establishing

co-referential links between other aspects if necessary. Thus in the mother-child-relation frame

```
(MOTHER-CHILD-RELATION
  (WITH SELF)
  (WITH MOTHER
        (THE PARENT OF A PARENT-CHILD-RELATION
             (WITH CHILD (= THE-CHILD))))
  (WITH CHILD (= THE-CHILD)))
```

Parent-child-relation is a generalization or superclass of mother-child-relation. Inheritance is guaranteed because of the co-referential links. The mother of every mother-child-relation is described as the parent of a parent-child-relation. Now suppose that we have another description attached to the parent of a parent-child-relation:

```
(PARENT-CHILD-RELATION
  (WITH SELF)
  (WITH PARENT (A PERSON))
  (WITH CHILD ))
```

then because this description, i.e. "(A PERSON)", holds for every parent of a parent-child-relation it holds *ipso facto* for the parent which is the mother of a mother-child-relation.

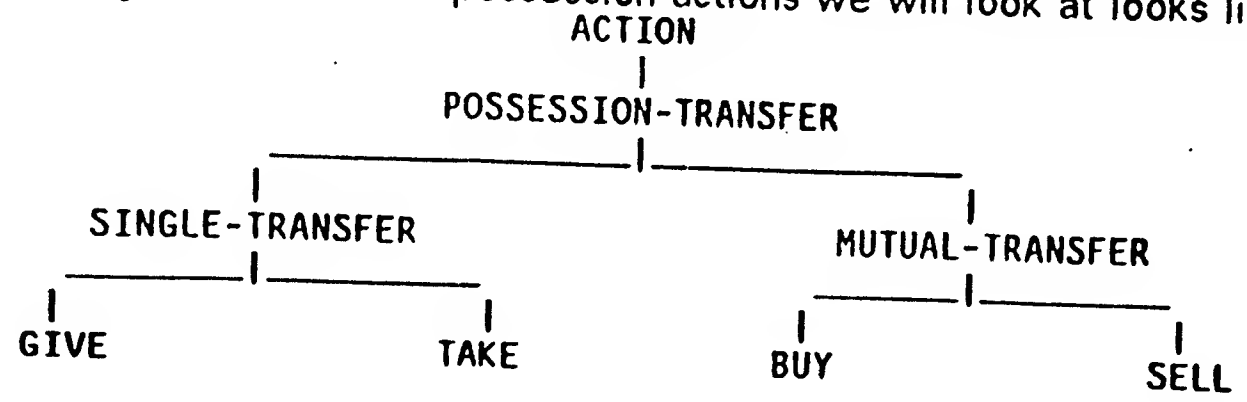
Again we have made certain decisions here that are not shared by other systems. In particular we see another application of the principle of modularity. A frame does never inherit aspects of another frame because that violates the locality assumption. An aspect would need to have meaning outside the frame in which it is defined.

Let us do another more extensive example to illustrate the representation of hierarchies. The example is concerned with a hierarchy of possession-transfer and has been discussed by other authors.

The subject of the example is transfer of possession. First we need a frame for possession itself. Possession involves an entity which has possession (the HAVER), an object that is being posessed (the OBJECT), and a situation in which this state of affairs occurs (the SITUATION):

```
(POSSESSION
  (WITH SELF)
  (WITH HAVER)
  (WITH OBJECT)
  (WITH SITUATION)).
```

The hierarchy of transfer-of-possession actions we will look at looks like this:



This hierarchy contains of course only a small subset of the transfer-of-possession

prototypes that might be in use, but the reader can easily imagine how to extent this hierarchy.

We discuss two types of possession-transfer: a situation where there is a single transfer: the old-owner no longer has an object and the new-owner gets the object, and a situation where there is a mutual transfer: the old-owner no longer has the object but gets something in return.
A single-transfer is further subdivided into three types: one where the old-owner performs the action, or is viewed as performing the action (GIVE), and one where the new-owner performs the action - maybe even without the old-owner knowing it (TAKE).

A mutual-transfer is worked out for the case where the exchange-object is money. Also here the action can be viewed as being performed by the old-owner (SELL) or by the new-owner (BUY).

We now construct frames for each of the nodes in the hierarchy that work out these specifications. The top-frame for ACTION has aspects for the action itself, for the actor, for the begin-situation and the end-situation:

```
(ACTION
  (WITH SELF)
  (WITH ACTOR)
  (WITH BEGIN-SITUATION)
  (WITH END-SITUATION)).
```

A transfer of possession can now be defined as being an action, such that in the begin-situation the old-owner has a certain object and in the end-situation the new-owner has the object :

```
(POSSESSION-TRANSFER
  (WITH SELF
        (AN ACTION
            (WITH ACTOR (= THE-ACTOR))
            (WITH BEGIN-SITUATION (= THE-BEGIN-SITUATION))
            (WITH END-SITUATION (= THE-END-SITUATION))))
  (WITH ACTOR)
  (WITH OLD-OWNER
        (THE HAVER OF A POSSESSION
               (WITH OBJECT (= THE-OBJECT))
               (WITH SITUATION (= THE-BEGIN-SITUATION))))
  (WITH NEW-OWNER
        (THE HAVER OF A POSSESSION
               (WITH OBJECT (= THE-OBJECT))
               (WITH SITUATION (= THE-END-SITUATION))))
  (WITH OBJECT)
  (WITH BEGIN-SITUATION)
  (WITH END-SITUATION)).
```

This structure can be paraphrased as follows. The action of a possession-transfer is an action whose actor is co-referential with the actor of the transfer, whose begin-situation is co-referential with the begin-situation of the transfer and whose end-situation is co-referential with the end-situation of the transfer. The old-owner of the

possession-transfer is described as the haver of a possession whose object is the object of the transfer and whose situation is the begin-situation of the transfer. Finally the new-owner is described as the haver of a possession whose object is the object of the possession-transfer and whose situation is the end-situation of the transfer.

Now we specialize into single or mutual transfer. Single transfer does not have to be a seperate frame because it would be the same as POSSESSION-TRANSFER itself. But for MUTUAL-TRANSFER we get

```
(MUTUAL-TRANSFER
   (WITH ACTOR)
   (WITH OLD-OWNER)
   (WITH NEW-OWNER)
   (WITH OBJECT)
   (WITH EXCHANGE-OBJECT)
   (WITH BEGIN-SITUATION)
   (WITH END-SITUATION)
   (WITH SELF
         (AN UNORDERED-COMPOSITION-OF-TWO-ACTIONS
                  (WITH ONE-ACTION
                        (A POSSESSION-TRANSFER
                                 (WITH ACTOR (= THE-ACTOR))
                                 (WITH OLD-OWNER (= THE-OLD-OWNER))
                                 (WITH NEW-OWNER (= THE-NEW-OWNER))
                                 (WITH OBJECT (= THE-OBJECT))
                                 (WITH BEGIN-SITUATION (= THE-BEGIN-SITUATION))
                                 (WITH END-SITUATION (= THE-END-SITUATION))))
                  (WITH OTHER-ACTION
                        (A POSSESSION-TRANSFER
                                 (WITH ACTOR (= THE-ACTOR))
                                 (WITH OLD-OWNER (= THE-NEW-OWNER))
                                 (WITH NEW-OWNER (= THE-OLD-OWNER))
                                 (WITH OBJECT (= THE-EXCHANGE-OBJECT))
                                 (WITH BEGIN-SITUATION (= THE-BEGIN-SITUATION))
                                 (WITH END-SITUATION (= THE-END-SITUATION)))))))
```

So, a mutual transfer is described as an unordered composition of actions where the first action is a possession-transfer where the old-owner is co-referential with the old-owner of the mutual-transfer and the object is co-referential with the object of the transfer. The second action is a possession-transfer where the old-owner is the new-owner of the transfer and the object is the exchange-object.

Use was made of an auxiliary frame for unordered composition of actions, i.e. actions that do not have to be in a specific ordering:

```
(UNORDERED-COMPOSITION-OF-TWO-ACTIONS
 (WITH SELF)
 (WITH ONE-ACTION)
 (WITH OTHER-ACTION)).
```

Next we specialize SINGLE-TRANSFER into two types: one where the action is performed by the old-owner (GIVE), and one where the action is performed by the new-owner (TAKE). We do this by attaching a description invoking an instantiation of

POSSESSION-TRANSFER to a slot in the GIVE-frame and establishing co-referential links to enable flow of information from GIVE to POSSESSION-TRANSFER.

```
(GIVE
 (WITH OBJECT)
 (WITH OLD-OWNER)
 (WITH NEW-OWNER)
 (WITH BEGIN-SITUATION)
 (WITH END-SITUATION)
 (WITH SELF
       (A POSSESSION-TRANSFER
             (WITH ACTOR (= THE-OLD-OWNER))
             (WITH OBJECT (= THE-OBJECT))
             (WITH OLD-OWNER (= THE-OLD-OWNER))
             (WITH NEW-OWNER (= THE-NEW-OWNER))
             (WITH BEGIN-SITUATION (= THE-BEGIN-SITUATION))
             (WITH END-SITUATION (= THE-END-SITUATION)))))).
```

Note how the actor of the possession-transfer is made co-referential with the old-owner. Similarly for TAKE:

```
(TAKE
 (WITH OBJECT)
 (WITH OLD-OWNER)
 (WITH NEW-OWNER)
 (WITH BEGIN-SITUATION)
 (WITH END-SITUATION)
 (WITH SELF
       (A POSSESSION-TRANSFER
             (WITH ACTOR (= THE-NEW-OWNER))
             (WITH OBJECT (= THE-OBJECT))
             (WITH OLD-OWNER (= THE-OLD-OWNER))
             (WITH NEW-OWNER (= THE-NEW-OWNER))
             (WITH BEGIN-SITUATION (= THE-BEGIN-SITUATION))
             (WITH END-SITUATION (= THE-END-SITUATION)))))).
```

Note how the actor of the possession-transfer is made co-referential with the new-owner.

A similar specialization of mutual transfer leads to a subdivision into BUY and SELL.

```
(BUY
  (WITH OBJECT)
  (WITH OLD-OWNER)
  (WITH EXCHANGE-OBJECT
        (AN AMOUNT-OF-MONEY))
  (WITH BEGIN-SITUATION (= THE-BEGIN-SITUATION))
  (WITH END-SITUATION (= THE-END-SITUATION))
  (WITH SELF
        (A MUTUAL-TRANSFER
              (WITH ACTOR (= THE-NEW-OWNER))
              (WITH OBJECT (= THE-OBJECT))
              (WITH OLD-OWNER (= THE-OLD-OWNER))
              (WITH NEW-OWNER (= THE-NEW-OWNER))
              (WITH EXCHANGE-OBJECT (= THE-EXCHANGE-OBJECT))
              (WITH BEGIN-SITUATION (= THE-BEGIN-SITUATION))
              (WITH END-SITUATION (= THE-END-SITUATION)))))
```

Note how the actor of the mutual-transfer is co-referential with the new-owner.

```
(SELL
  (WITH OBJECT)
  (WITH OLD-OWNER)
  (WITH EXCHANGE-OBJECT
        (AN AMOUNT-OF-MONEY))
  (WITH BEGIN-SITUATION)
  (WITH END-SITUATION)
  (WITH SELF
        (A MUTUAL-TRANSFER
                (WITH ACTOR (= THE-OLD-OWNER))
                (WITH OBJECT (= THE-OBJECT))
                (WITH OLD-OWNER (= THE-OLD-OWNER))
                (WITH NEW-OWNER (= THE-NEW-OWNER))
                (WITH EXCHANGE-OBJECT (= THE-EXCHANGE-OBJECT))
                (WITH BEGIN-SITUATION (= THE-BEGIN-SITUATION))
                (WITH END-SITUATION (= THE-END-SITUATION))))).
```

Note how the actor of the mutual transfer is co-referential with the old-owner.

The exchange-object was in both cases restricted by reference to a frame for a certain amount of money:

```
(AMOUNT-OF-MONEY
  (WITH SELF)).
```

So far we have constructed the hierarchy such that a bottom-up flow of information is possible, i.e. if we know that something is a buy-action, we know that it is a mutual-transfer and thus we inherit the fact that it is a transfer-of-possession and an action. Now we add descriptions to allow a flow of information in both directions. This is done using conditional descriptions.

We will attach a description to a frame higher up in the hierarchy indicating what frames are specializations of the concept and on what basis the specialization is made. The result is a sort of discrimination network that can be searched based on conditional descriptions. One example is sufficient to give an idea.

Suppose we have the possession-transfer frame given earlier and we want to create downward links to give and take. What we do is attach a conditional expression to the self-slot, saying that when the actor of the action is equal to the old-owner, the action is a give action, and when the actor of the action is equal to the new-owner, the action is a take action:

```
(POSSESSION-TRANSFER
  (WITH ACTOR)
  (WITH OLD-OWNER)
  (WITH NEW-OWNER)
  (WITH OBJECT)
  (WITH BEGIN-SITUATION)
  (WITH END-SITUATION)
  (WITH SELF
        (IF THE-ACTOR IS
              ((= THE-OLD-OWNER)
               (A GIVE
                  (WITH OLD-OWNER (= THE-OLD-OWNER))
                  (WITH NEW-OWNER (= THE-NEW-OWNER))
                  (WITH OBJECT (= THE-OBJECT))
                  (WITH BEGIN-SITUATION (= THE-BEGIN-SITUATION))
                  (WITH END-SITUATION (= THE-END-SITUATION))))
             ((= THE-NEW-OWNER)
              (A TAKE
                  (WITH OLD-OWNER (= THE-OLD-OWNER))
                  (WITH NEW-OWNER (= THE-NEW-OWNER))
                  (WITH OBJECT (= THE-OBJECT))
                  (WITH BEGIN-SITUATION (= THE-BEGIN-SITUATION))
                  (WITH END-SITUATION (= THE-END-SITUATION)))))))
```

DISCUSSION

The standard way of developing concrete theoretical proposals in AI is to construct a language which contains primitives that embody concepts advanced as necessary and sufficient by the theory. If it is discovered by experimentation, i.e. by trying to construct reasoning systems for particular problems, that certain constructs are not necessary or that the constructs are not sufficient - or even worse harmful - the language is abandoned and a new stage of development is entered.
A nice example of this methodology can be seen by looking at the argumentation for going from PLANNER (Hewitt, 1969) to CONNIVER (McDermott and Sussman,1972) in McDermott and Sussman (1973).

This methodology is also used here. In the previous paragraphs we have introduced a particular language that embodies primitives for representing knowledge according to the principles that have been proposed earlier on.

The concept of a frame was introduced by Minsky (1974) in an attempt to construct a theory based on the principles presented in the first section. His proposal triggered the development of a whole series of so called frame-based languages that tried to elucidate this notion. These languages can be classified into three categories:

The first category used statements (in the form of patterns) as basic representation tool to assemble frames. Examples are MDS (Srinivasan,1976) and the frame language developed by Charniak (1977). The use of patterns reflects earlier work on pattern-directed invocation systems (cf. Hewitt,1968) and the assumption is that techniques developed in such systems (like pattern-matching and invocation) would be used for implementing the reasoning system.
The second category is rooted in the tradition of semantic networks: a particular data structure is proposed and a number of access and construction functions are defined. Major examples of this

approach are FRL (Roberts and Goldstein,1976) and the language proposed in Kuipers (1977). The third category to which the present language belongs is based on the idea of a description. This idea was first explored in KRL (Bobrow and Winograd, 1977), which is a very baroque language in the sense that it contains features of the three categories. Nevertheless there are many similarities between the language presented in this chapter and KRL.

It is also important to see the relation to so called constraint languages, like Sussman and Steele (1979) or Borning (1979), where a body more or less corresponds to a frame, a cell to a slot, a constraint to a description, a merge to a co-referential description. These constraint languages assume however that there is a unique way of describing the answer to a particular question posed by a frame (e.g. in terms of a numerical value) which makes them more restricted than a general description language. At the same time these languages do not allow for (what they call) meta-constraints which are here captured by the conditional description.

There are two other major paradigms for knowledge representation: predicate calculus and semantic networks. The difference between frame-based knowledge representation and predicate calculus is simply that predicate calculus violates all the principles we have proposed in section 1 of this chapter. There is no modularization, no organization, no hierarchy and a bias toward statements rather than descriptions (predicate-functor logic is an exception to this). Moreover other principles that will be discussed in upcoming sections will also be shown to be violated by predicate calculus. This does not mean however that the representation schemes are incomparable. In fact it would be a good exercise for the reader to construct a mapping from the present language to predicate calculus. See for arguments for and against predicate calculus Hayes (1974,1977) and Moore (1979) vs. Minsky (1974) and Hewitt (1975).

One of the main differences between semantic networks and frame systems is that there is no modularization in a semantic network. Recently however most network designers (Hawkinson (1979), Levesque (1977), a.o.) have proposed input-languages. These languages make use of modules and use object-oriented descriptions. Although this is an important step toward modularization, it must be noted that if the modularization is not maintained in the internal representation (i.e. has no effect on the access-functions) the modularization principle is violated. On the other hand, partitioning (as described in Hendrix,1975) could be used to introduce a frame-like organization in a semantic network (cf. Scragg, 1975).

The examples made use of an important principle of conceptual analysis first advanced by McCarthy (1958) and McCarthy and Hayes (1968). This principle is known as the situational calculus and consists in attaching situational tags to each predicate that is situation-dependent. The possession-transfer example is a classical example and has been treated by many authors, see e.g. Fillmore (1977), Schank (1975), a.o.)

EXPERTS

3. EXPERTS

The previous chapter was devoted to problems of representation. We addressed the question how knowledge structures to be used for constructing models of problem situations should be represented and organized. We now turn to the counterpart of this problem: How should the resulting knowledge structures be activated. This topic can be further divided into two subtopics: (i) what are the computational primitives and the architecture of the activation mechanisms, and (ii) what are *in concreto* the actions that are taken while reasoning about a particular problem. In this chapter we take up the first question. In particular, we will try to find out what possible computational architecture we will need. Just as in the first chapter this chapter is divided into two subsections: one discussing a series of principles that will underly the specific design proposals and one discussing the system itself.

1. THE PRINCIPLES

1. 1. THE ISSUE OF INDEPENDENCE

There are at least two ways in which a model of the problem situation can be maintained. The first way is familiar from ordinary programming: when one writes a program to perform a particular task, this program keeps parts of the problem situation in special places (variables) and the subprograms are carefully constructed so that they know where to go and look for the information when they need it. Such a program does not maintain an *independent* representation of the model. The alternative is that the model is a representational object that exists independently of the processes constructing and consulting it. Such a representational object or datastructure is often called a database. This alternative will be adopted here based on the following argument.

In most problem solving situations where the method of solution is unknown in advance, a process that is able to create a datum usually does not know what other processes might need it. The only thing it can do is make the new fact available to others.
On the other hand a process usually does not know where to go and look for its data. Does a certain fact needed to understand the line in a story come from the previous text, an inserted figure, or things you are supposed to know?
The notion of an independent model nicely solves these problems: If a process finds a fact it adds it to the model, if a process needs a fact it looks into the model. Hence the following principle:

PRINCIPLE 7:
THE MODEL BUILT UP BY THE REASONER IS AN INDEPENDENT OBJECT.

This means concretely that if all processes are finished, the model is still there, or that an 'outside observer' can look at a model even if this observer is not one of the processes initially involved in its construction.

1. 2. THE ISSUE OF MODULARIZATION

Based on this principle a model can be viewed as a database of facts, where each fact is a statement saying that a particular description holds for a particular object. But now we observe that there is another complexity problem. Even for moderately simple problem situations there will be thousands of facts. Each of these facts might be relevant at any point in the reasoning process. So how do we guarantee efficient access?

The answer is modularization: all descriptions which hold for one object could be collected in a unit. When the reasoner then wants to access a particular fact, it can look in the units of the objects that contain this fact. For example all descriptions which hold for John are collected into a unit. When something is said about John, this is stored in the unit for John and when a fact needs to be looked up where John is involved, the reasoner looks into the unit for John. Such a unit therefore serves as the terminal of the various slots in the frames in which the individual plays a role. We call this type of modularization *object-oriented model organization.*

It is clear that we gain enormously in efficiency if we follow this principle. Even when there are thousands of facts, the problem of searching through them becomes almost trivial because we can access a description partly based on its content, i.e. on the objects that play a role in the description. Thus the access to descriptions will not significantly degrade when more objects are introduced.

This is summarized in the following principle:

PRINCIPLE 8:

A MODEL CONSISTS OF A COLLECTION OF UNITS, WHERE EACH UNIT CONTAINS ALL DESCRIPTIONS WHICH ARE TRUE FOR A PARTICULAR OBJECT IN THE MODEL.

Such a unit might be thought of as a frame that contains descriptions about a particular object in the model of a particular problem situation.

1. 3. THE ISSUE OF MODULARITY CONTINUED

The 'substance' that was studied so far consisted of passive, representational objects: frames, descriptions, procedures. It is now time to turn to the active part of a reasoning system. A useful concept that will enable us to talk about this is the notion of a *process.*

A process contains a procedure that defines what actions will be performed and a local environment containing information that will be needed for the execution of the procedure or will be affected by it. A procedure with its environment is often called a *closure.* The closure becomes a process when the procedure is executed in the environment. In the literature the word process is often used to mean potential process, i.e. closure, as well.

Now we observe that the complexity problem noted as regards the amount of knowledge available, holds *ipso facto* for the processes that will be needed to activate knowledge. If we have to deal with thousands of pieces of knowledge, we will need thousands of processes that compare these pieces and perform operations to obtain new knowledge. The problem is how do we guarantee that all closures will be executed? Because there are so many of them we do not want a closure to wait an arbitrarily long time before it becomes active, in fact we want a closure to be executed as soon as possible. A second problem is that we do not want the whole system to come to a halt because something went wrong in one part, i.e. damage should not be fatal but limited to the restricted area in which the damage occurs.

Observe that, interestingly enough, biological organisms have similar problems. In an organism like the human body thousands of reactions, i.e. processes, have to be performed - even for simple maintenance. There must be some sort of guarantee that the reactions will be performed and each reaction requires its own delicate balance of materials. Biological organisms use a powerful solution to deal with this complexity: *modularization*. Reactions related to a particular task are put together in units. At the micro-level such a unit is called a cell, at a macro-level an organ.

We will do the same thing here: processes will be organized in units, so that processes in a unit are 'protected' from what goes on elsewhere. And if we give each unit the power to execute its own processes, then we solve the other problem as well, namely that a process is executed as soon as possible after its closure is formed.

Note how we move here away from a sequential mode of execution which would imply that there is only one locus of computation often called the central processing unit and that every process is performed one after the other by this unit. In contrast we have come to adopt the principle of parallel computation which assumes that there are many places where computation could take place and processes could run independently in all these different places.

The final issue we have to consider in this context is on what principle the modularization will be based. Because information in the knowledge-base and in the model is already organized in terms of the conceptual structure of the domain it makes sense to follow the same organization for the processes. A nice consequence of this is that the processes will be close to the descriptions they will need or to the unit in which they have to add descriptions. So we have come to the conclusion that each of the units we postulated earlier on is an active unit, i.e. contains a number of processes that are busily working on the descriptions contained in that unit. This is expressed in the following principle:

PRINCIPLE 9:

REASONING PROCESSES ARE LOCATED IN ACTIVE UNITS REFLECTING THE CONCEPTUAL STRUCTURE OF THE DOMAIN OR THE MODEL.

Based on this principle we arrive at a new theoretical unit that has three aspects:

(i) it contains knowledge about a particular (limited) subject-matter, i.e. a

frame,

(ii) it contains a set of (potential) processes related to the subject-matter covered by the frame, and

(iii) it contains a locus of computation (the necessary energy so to speak) to evaluate its closures.

We call this new unit *an expert* because it knows everything that is known about a limited subject-matter. Experts will be organized in terms of *societies*, where a society is the procedural analogue of a micro-world. Because frames are hierarchically organized in a microworld, the experts in a society will also be hierarchically organized. Using the biological metaphor we could say that an expert corresponds to a cell and the society to the organism.

Let us now explore further constraints on these new theoretical units. In particular we investigate how interaction takes place, whether there are any constraints on possible interactions and how experts are formed.

1. 4. THE ISSUE OF PROTECTION

Concerning the interaction between experts we see two alternatives: Either every expert is allowed to do whatever it wants directly. For example, if an expert wants to know something about another expert it simply looks inside that other expert, if an expert wants to change the state of another expert, it changes the state, etc.

The alternative is to give every expert relative autonomy so that it can protect itself from possibly undesirable actions by other units, i.e. one expert would not be allowed to treat another expert as an object. The way to realize this idea is to postulate that all interactions between experts must go by message passing: If an expert wants to see anything done, it should send a message so that the other expert can decide for itself whether it will accept the task of doing it.

There are some important technical reasons for adopting this second more cautious view. One illustrative example problem is the so called airline reservation problem. If there is one seat left on the plane and two travel agents both grab the seat at the same time (because they both checked at the same time and the seat was empty), who is going to get the seat? With a parallel process organization one needs greater protection because a certain process might change the data of another process even though this other process checked just a moment ago and therefore thinks the data are still there.

Another reason to give an expert relative autonomy is to protect delicate structures from being changed all the time. This problem will not come up in the present work but it is - we believe - a crucial issue when learning is considered. Suppose one person tells you something and you believe him, then you will at least show some resistance if a little bit later another person tells you that the opposite is true. If we would instantly change our knowledge structures whenever something was told to us, our knowledge would show no coherence and one module would not be able to rely upon another module.

This leads us to adopt the following principle:

PRINCIPLE 10:
INTERACTION BETWEEN EXPERTS IS PERFORMED BY MESSAGE PASSING.

1. 5. THE ISSUE OF LOCALITY

If all interaction is performed by message passing the next issue becomes whether there are any constraints on communication. There are two possibilities: Either everybody is allowed to communicate with everybody else, i.e. there are some general broadcast mechanisms or methods to establish arbitrary communication links, or there are very tight constraints on possible communications, i.e. there is only local interaction.

We will argue in favor of locality. Recall that one of our major problems is complexity. In previous paragraphs we introduced various measures to do something about this, in particular we proposed to modularize and organize the space of knowledge and processes into units reflecting the conceptual structure of the domain. It is clear that if we would now accept global interaction between all the units the modularization principle would be seriously violated. That is why we adopt the locality thesis:

PRINCIPLE 11:
EXPERTS CAN ONLY COMMUNICATE WITH A LIMITED NUMBER OF OTHER EXPERTS.

More specifically, because experts are organized in hierarchies, we will only allow communication between experts that are hierarchically related. Thus experts dealing with aspects of a certain prototype can communicate with the expert dealing with the object as a whole, etc. Nonlocal message passing like a general broadcast are excluded.

It should be clear that these constraints on possible communication are very strong. They are so strong in order to prevent the system to come to a virtual halt because of excessive communication.

1. 6. FORMATION

The final issue that will be raised here concerns the origin of the experts. There are a number of potential solutions to this question but this is the one that seems most promising.

An expert could come into existence as a copy of another expert which acts as its prototype. In other words given an expert with a body of knowledge about a particular subject-matter and a set of (potential) processes, it would have the capability to construct a copy of itself and let this copy work on a certain problem that otherwise the expert itself would have to work on. In practice this copy will be a virtual copy, i.e. portions of the expert which remain the same are physically the same and it is only when

differences occur that new objects are created.

The main argument for doing things this way is orderly organization of the activities and economy of the system. If the same expert needs to keep track of many different problem situations, we would need horribly complicated bookkeeping mechanisms.

We can now see also how hierarchies might be formed. Assume that at a given moment of time there is some expert having a certain amount of expertise, i.e. a number of descriptions organized in a frame and a series of (potential) processes. Suppose furthermore that this expert is able to do something (maybe to its own surprise) and experts in the environment of the expert notice this fact. For example, a particular sensori-motor action, being looked at by a certain expert, is performed and has by accident an interesting effect. If such a thing happens we postulate that the current state of the expert is "frozen" and experts present at this moment make a record of what this expert can be used for.
Next time a similar activity or an instance of the same object comes up the prototypical expert can create a copy of itself to deal with the new instance. This new copy might again discover new things about the particular instance it is working on, be preserved, etc. This is the way the hierarchy constituting a society might get formed.

This is reflected in the final principle:

PRINCIPLE 12:
EXPERTS START OFF AS COPIES OF OTHER EXPERTS.

In this section, we proposed a number of constraints on a possible architecture for a reasoning system. In particular, we proposed to modularize and organize processes along the same lines as is the knowledge itself. This lead us to adopt a new unit called an expert which is an active object that groups a number of processes. An expert comes into existence as a copy of another expert acting as its prototype. Various constraints were introduced on what this object can do: one expert cannot treat another expert as an object, interaction goes by message passing, an expert can only communicate with a limited set of other experts and experts operate in parallel.

From this section, first glimpses could be seen of the major metaphor we propose in this work: reasoning modeled as a society of communicating experts. Before it will be fully clear what is meant by this, we have to explore many other topics.

DISCUSSION

The idea that models have to be independent is the central thesis of pattern-directed invocation systems. The argument for the principle is due to McDermott and Sussman (1974).

The idea that frames should be turned into active objects which operate by exchanging messages emerged soon after the frame-hypothesis was formulated, i.e. around 1975. Realizations go under

the name of AGENTS (Minsky and Papert,1976), ACTIVE SCHEMATA (Bobrow and Norman,1976), BEINGS (Lenat,1977), STEREOTYPES (Hewitt,1975), etc. Our notion of an EXPERT is meant to capture the essential properties of these closely related concepts.

From a certain viewpoint the productions in a production system (as in Newell and Simon,1972) can be compared to the independent active objects we talked about. However there are important differences which cannot all be discussed here. For example production systems operate on a global data base, whereas here every object has its own database and others are not able to look at it explicitly, etc. Production systems usually also assume a sequential mode of operation, whereas we have argued for parallelism.

Some of the principles we have proposed here have been influenced by developments in computation, especially concerning so called 'message passing systems', emerging from Smalltalk (Kay,1976) and the ACTOR-theory of computation developed by C. Hewitt and his associates (Hewitt, et.al.(1973), Hewitt (1976), Baker (1978), Yonezawa (1978), a.o.). This theory currently takes the form of a mathematical framework that is attractive as a foundation for the reasoning system because it reflects many of the principles we introduced earlier on: (i) modularity and distribution of knowledge, (ii) one unit cannot treat another unit as an object, (iii) basic interaction is message passing, and (iv) locality.

The dominant paradigm for process organization until very recently has been sequential. This was not necessarily because most computers were based on a serial process-organization. Theorists aruged explicitly for sequentialism (see e.g. Newell and Simon (1972)). Arguments in favor of parallelism for problem solving can be found in Fennell and Lesser (1975) and Kornfeld (1979).

2. THE FRAMEWORK

We now introduce a conceptual framework based on the principles presented in the previous paragraphs. Then we will propose a concrete system which will be the basis for mechanizing the reasoning behavior that will be proposed in the next chapter.

2. 1. EXPERTS

The new fundamental unit of the system is the expert. An *expert* is defined to be an active object that has a body of knowledge about a particular subject-matter (in terms of a frame) and a script.
The *script* contains a number of rules, i.e. closures, which specify how the expert should behave: how it should respond to requests for information about the knowledge it is responsible for, how it should try to expand incoming descriptions by asking questions to other experts, how it should maintain consistency of its descriptions by preventing the introduction of contradictions, etc. Some of the rules may be active, others may be waiting for messages coming in.
Both the body of knowledge and the script are dynamic entities: they can change and grow depending on the functioning of the whole system and the structure of the tasks they have to deal with.

EXPERTS

The experts obey all the principles discussed earlier:
+ Experts operate in parallel.
+ An expert cannot treat another expert as an object, i.e. all interaction is by message passing.
+ An expert can only communicate with a limited number of acquaintances.
+ Experts are hierarchically organized due to the way they are created.

The 'body of knowledge' of an expert consists of a collection of descriptions possibly embedded in control-indicators. The following is an example

```
(you-are-described-as (THE MOTHER OF A FAMILY)).
```

Each of those descriptions enters the expert as a message.

The 'rules' in the script contain a condition and an action. The condition is a predicate which holds for a message that is sent to the expert. The action says how the expert should respond to a message of the type described by the condition. It often happens that the expert needs to wait for a message, in which case the rule is stored and tried out on all messages until the one that matches comes in. It can also happen that the condition concerns a description which arrived as a message prior to the formation of the rule. In this case the expert compares the condition with the descriptions in its database and will execute the action if the condition is satisfied. In these two cases a rule in the script of an expert is like a pattern-directed invocation rule.

Experts themselves are fairly complicated objects. We will now describe their behavior in more detail. We will do that using the ACTOR-theory of computation as framework. This will show that there is not only parallelism between the various experts, but also inside an expert.

An expert is an actor with two basic acquaintances: a database and a rule-set.

An expert has the following script:
1. It may receive a message to add a new description to its database, in which case it sends this new description to its database.
2. It may receive a message to add a new rule to its database, in which case it sends this new rule to its rule-set.
3. It may receive a message to tell more about a description with certain specifications in which case the request is sent to the database-actor and the reply will be forwarded to the expert requesting the information.
4. It may receive a message to make a copy of itself and to send a particular task to this copy.

The database of an expert is another actor which has as acquaintances a number of descriptions.

The database-actor has the following script:
1. It may receive a message telling about a new description, in which case (a) the description is added so that the database-actor will be able to respond to questions about it later, and (b) a message is sent to the rule-set of the expert telling that a new

EXPERTS

description has been added.

2. It may receive a request to check whether a description with certain properties is among its descriptions, in which case an answer is returned telling whether the description is in the set or not and if so telling the known properties of the description.

The rule-set is an actor which has as acquaintances a number of rules. Each rule is in turn an actor with two acquaintances: a so-called condition, which describes the properties of a description, and a body, which contains a script to be executed when a description in the database-actor is found that has all the properties described in the condition.

A rule-actor has the following script:

1. It may receive a message telling to become active in which case it asks the database-actor whether it knows of any description that meets its condition. If so the script is executed, which means that new messages will be sent to some expert.

2. It could receive a more specific request telling to confront its condition with a certain description in the database. If the description meets the specifications of the condition, the script is executed.

The rule-set actor has the following script:

1. The rule-set-actor may receive a message telling about a new rule, if so the rule is added to the existing stock of rules, and a message is sent to the actor to make it become active.

2. It may receive a message that a new description has arrived in the database-actor in which case every rule in the rule-set will be sent a message to become active.

Each expert has only a limited set of other experts as acquaintances, namely those of which it knows the name (which may be available in the scripts or could be told by the other expert) and its ancestor (i.e. the expert that sent the message to create the expert). Also an expert is acquainted with itself.

A metaphor may help to visualize this communication network. One can think of an expert as having a telephone and a telephone book with a very few numbers. An expert can only call the experts it has the number of. As long as no other expert calls up, other forms of communication are impossible, and even in this case the expert cannot call back unless the other expert has given its number. Communicating with an expert which functions like a prototype is similar to calling an information service (such as for airline reservation). Here the person calling knows the number of the company but does not know which person (or machine) will actually respond to its request.

2. 2. CONVENTIONS

In order to make all these proposals amenable to concrete experimentation, we have designed and implemented a language which contains primitives for constructing reasoning systems along the lines advocated in previous paragraphs. We will now present some of the main functions in this language

Evaluation of the form

(CREATE-EXPERT <name-of-expert>)
creates an expert which is equipped with mechanisms to maintain a database of descriptions, schedule and execute parts of its script, etc.
The <name-of-expert> can be a particular name, e.g. XPRT-1 or CHAIR, in which case the expert from then on has the given name, or it can be the name of an already existing expert preceded by an indefinite article, as in
(CREATE-EXPERT (a CHAIR)).
In this case the expert with the given name functions as a prototypical expert and the resulting expert will have a unique new name and all the properties the prototypical chair had at the moment of creating this expert.

Evaluation of the form
(TELL <name-of-expert> <description>)
sends a description to an expert with name <name-of-expert>. The result is that after the arrival of the message, the expert will include the description in its database.
<Name-of-expert> can be the explicit name of an expert or the name of a prototypical expert preceded by an indefinite article. The latter way of calling up an expert will result again in the creation of a copy of the prototypical expert and in the action of sending this new copy the message. An expert can also send a message to itself in which case the name of the expert would be :MYSELF.

Now we turn to mechanisms for installing a rule in an expert. There are three types, mirroring the conditional-descriptions we introduced in the description language:
(i) A rule that will check at the moment of its creation whether its condition is met by a description in the database. If that is not the case, it executes the action it has for the case when the condition is not met. Then the rule aborts itself. If it is the case it executes the action it has for when the condition is met and the rule aborts itself.
(ii) A rule that will check at the moment of its creation whether its condition is met by a description in the database. If this is the case the rule executes the action it has for when its condition is met and aborts itself. If the condition is not met by any description, the rule patiently waits for other messages coming in which contain a description that might satisfy its conditions. When such a description comes in the action is executed and the rule aborts itself.
(iii) A rule that will check at the moment of its creation whether its condition is met by a description in the database. If this is the case the rule executes the action it has for when its condition is met but keeps on looking out for other descriptions that match its condition. For each case where there is a match the action is performed. When there is no description that satisfies its condition at the moment the rule is created, it will continue waiting and will perform an action each time a match occurs.

Here are some syntactic forms for expressing rules of each type. The condition of these rules is each time a description.

(INSTANTANEOUS-ASK <name-of-expert>
<condition> <action-if-success> <action-if-failure>)
installs a rule of the first type in an expert with name <name-of-expert>. The <name-of-expert> can again be three things: the expert itself, in which case the expert

looks whether it knows about a certain description and executes an action if this is the case, the name of a particular (prototypical) expert, (e.g. an expert may ask the CHAIR-expert whether chairs have legs), and finally the name of a prototypical expert preceded by 'a' or 'an' which causes the creation of a copy which is then sent the actual message.
The <action-if-failure> is optional.

For example,

```
(INSTANTANEOUS-ASK XPRT-1
                   ;; the condition:
                   (IS (A TABLE))
                   ;; the action if match occurs:
                   (TELL XPRT-2 (XPRT-1 IS (A TABLE)))
                   ;; the action if no match occurs:
                   (TELL XPRT-2 (XPRT-1 IS (NOT (A TABLE)))))
```

When at the time of evaluating this form XPRT-1 contains the description

```
(IS (A TABLE))
```

as one of the descriptions in the database, a message will be sent to XPRT-2 asking it to add the description that XPRT-1 is (A TABLE) to its database. When XPRT-1 does not contain the description

```
(IS (A TABLE))
```

XPRT-2 will be sent a message asking it to add the description that XPRT-1 is (NOT (A TABLE)) to its database.

There are some complications in the sense that the condition may contain variables that have to be bound to specific values and these bindings are to made known to the action of the rule. There are two types of variables. Variables which will match with parts of a description and co-referential descriptions. The first type will be represented by putting a colon in front of the atom denoting the variable as in

```
:VAR
```

One variable is already known for the start: :MYSELF which is bound to the name of the expert containing the rule.

```
(SINGLE-EVENT-ASK <name-of-expert>
                  <condition>
                  <action-if-match-occurs>)
```

installs a rule of the second type. The name-of-expert can be the same thing as with the previous form.
For example,

```
(SINGLE-EVENT-ASK XPRT-1
                  (IS (A TABLE))
                  (TELL XPRT-1 (IS (AN OBJECT))))
```

Here XPRT-1 will look out for a description of the form

```
(IS (A TABLE))
```

and once this description is found it will send to itself a message requesting to add the description

```
(IS (AN OBJECT))
```

to its database of descriptions.

Finally,

```
(CONTINUOUS-ASK <name-of-expert>
                <condition>
                <action>)
```

will install a rule of the third type. The expert with the given name will continuously look out for a description matching its condition and will perform the action each time it is the case.

These are some of the main operations of the system that have been built to simulate the architecture implied by the model developed in this chapter. There are some other functions which allow us to look into an expert, trace message passing, perform certain types of initialization, etc. Also it is possible to give continuous input. But attention to these details now would distract from the main lines of this work.

2. 3. EXAMPLES

The following examples may give the reader some idea of the message-passing behavior that is made possible by these primitives.

The system prompts with a >>-sign. The following action creates a new expert called TABLE.

```
>> (create-expert TABLE)
```

The next example illustrates a TELL action. The expert prints out the messages it received.

```
>> (tell TABLE (IS (A TABLE)))
===> TABLE receives the following message from USER:
(IS (A TABLE))
```

The following is an example of a SINGLE-EVENT-ASK. The expert will find a description corresponding to its condition and execute the resulting action, which is a new TELL action.

```
>> (SINGLE-EVENT-ASK TABLE (IS (A TABLE))
        (tell TABLE (IS (AN OBJECT))))
===> TABLE receives the following message from TABLE:
(IS (AN OBJECT))
```

Now follows an example of the creation of a copy of a prototypical expert. The copy (called DESK-1) gets the message which would otherwise go to the prototypical expert itself:

```
>> (tell (a TABLE) (IS (A DESK)))
===> DESK-1 receives the following message from USER:
(IS (A DESK))
```

The next example illustrates INSTANTANEOUS-ASK. This example also demonstrates that DESK-1 inherits all properties from the TABLE expert. We never told DESK-1 explicitly that it has a table-top and nevertheless the description triggers an action.

```
>> (INSTANTANEOUS-ASK DESK-1 (IS (A TABLE))
          (TELL :MYSELF
                (IS (THE TABLE OF A ROOM))))
===> DESK-1 receives the following message from DESK-1:
(IS (THE TABLE OF A ROOM))
```

The next transmission adds a new rule to the script of TABLE but nothing happens because the description is not part of the descriptions this expert has.

```
>> (SINGLE-EVENT-ASK TABLE (IS (THE OBJECT OF A SQUARE-SHAPE))
        (TELL :MYSELF (IS (A SQUARE-TABLE))))
```

The following example illustrates that the script is inherited by each copy of a prototypical expert. Because the copy of table receives a message that will trigger one of the rules inherited from the prototypical expert it executes the action with a new message-exchange as a result.

```
>> (tell (a TABLE) (IS (THE OBJECT OF A SQUARE-SHAPE)))
===>  OBJECT-1 receives the following message from OBJECT-1
(IS (A SQUARE-TABLE))
```

This should give the reader some idea of the message passing behavior that is allowed by the constructs proposed in this chapter. In the next chapter we will show how a reasoner can be constructed with these primitives.

DISCUSSION

There are two lines of development leading to the language introduced in this section: pattern-directed invocation languages and message-passing systems.

The development of pattern-directed invocation languages was triggered by PLANNER (Hewitt, 1969) and resulted in a whole series of AI-languages. Some of the major examples are CONNIVER (McDermott and Sussman,1972), QA4 (Rulifson, et.al,1973), POPLER (Davies,1973), AMORD (deKleer, et.al.,1977), ETHER (Kornfeld,1979). The relation between these languages and the present language is such that the list of facts inside an expert can be viewed as a (tiny) database of patterns and the rules in the script can be viewed as pattern-directed invocation rules. The description that the database is modularized leads to a great efficiency in the lookup of patterns .

The other related development is made up by so-called message-passing languages. The first language based entirely on message-passing is SMALLTALK (Kay,1976). Other languages are PLASMA (Hewitt and Smith,1975), DIRECTOR (Kahn, 1978) and ACT1 (Hewitt and Lieberman,1979). These languages have most of the primitives we postulated as necessary and sufficient and it should therefore be straightforward to implement the reasoning system we envisage in one of them. We have designed and implemented our own language mainly with the purpose of having a basic and therefore efficient message passing system that is nevertheless sufficient for our purposes.

REASONING

4. REASONING

Chapter 2 discussed ways of specifying knowledge about a particular domain. Chapter 3 contained specific proposals for the computational architecture of reasoning systems. This chapter is devoted to bringing these two lines of development together in order to construct an actual reasoning system. Before we present in detail the reasoning behavior adequate for the language in chapter 2 we will discuss first some issues that will partly determine what this reasoning behavior will look like.

1. THE PRINCIPLES

In previous chapters we already introduced many important principles that determine how reasoning will proceed. For example, it has been proposed that reasoning involves the construction of a model of the problem situation, that this model is to be an independent object that contains descriptions of the objects from multiple viewpoints, that it will consist of a collection of experts that perform the construction by exchanging messages, etc. There is little that we have to add to all this, except one additional important principle.

1. 1. THE MODE OF OPERATION

At issue is whether the construction should be goal-directed, i.e. be driven by the nature of the solution of a particular problem and only working out parts of the model in as far as they bring us closer to a solution, or whether the reasoner should expand the model as complete as it can, i.e. develop all consequences of the initial specifications so that possible goals will eventually be realized.
The first mode of operation is often called backward chaining or *consequent reasoning* because the actions are triggered by what one wants to obtain as result. The second mode of action is often called forward chaining or *antecedent reasoning* because the actions are triggered by the initial constraints of the problem situation.

It would be nice if we could let the construction of a model be determined completely by the goals the system has at a particular point in time. However this turns out to be an untenable position. We observe for example that if we have a hierarchy of animals with probably thousands of specializations attached, then we would have to consider an arbitrary sample each time we want to show that something is an animal.
But pure antecedent reasoning is equally bad. All knowledge about a particular person will be accumulated, for example, even though we might only be interested in his age.

Nevertheless we will assume that antecedent reasoning is the basic mode of operation. We can do this based on two conditions

(i) The goals themselves should be explicitly specified so that deduction can be based on explicit goals (this topic will be explored in chapter 6)
and

(ii) The model is organized into a finite structure based on the idea that one can

distinguish a finite set of important questions about the problem situation. This finite structure is sometimes called a *constraint network* and the antecedent-reasoning in such a network is often called *propagation of constraints.*

The reasoning process can now be described as follows. The system starts off by absorbing data from the problem specification. These data will cause the accumulation of an initial set of answers, i.e. constraints, for the various questions. These constraints trigger various constraint relationships which will cause the (parallel) accumulation of new constraints by local one-step deductions eventually leading to a complete analysis or synthesis. This accumulation process is also known as *progressive refinement* because the answer to a particular question is gradually refined by incoming descriptions.

The important thing about this constraint propagation methodology lies in the approach towards the control structure problem. Rather than have one viewpoint of analysis be responsible for controlling where computational resources will go, or have a general supervisor that overlooks the analysis and allocates resources according to some algorithm embodying a certain problem solving strategy, the constraints themselves are put in charge of control. They will take resources when they feel that the current problem situation supports the constraint, or refrain from operating when this support is not or weakly available.

This leads us to the following principle:

PRINCIPLE 13:

THE PRIME MODE OF OPERATION IN THE CONSTRUCTION OF A MODEL IS PROPAGATION OF CONSTRAINTS.

Note that all this fits nicely with what was developed so far. A frame is a series of important questions that should be asked about a particular (limited) subject-matter. It can therefore serve as the basis for developing the constraint network, which is the global set of questions we want to ask about the problem situation. We also see now that the experts which constitute the model are actually a constraint network. Each node in the network corresponds to an expert. Propagation itself will be phrased in terms of message passing. When a certain description arrives at an expert its consequents will be propagated to the other experts that are involved in the instantiation implied by the description. A useful metaphor to describe the reasoning process is that of a *question-answering activity*: the frames bring up questions and the experts try to accumulate as much constraints as possible on the answer to the question it is responsible for.

DISCUSSION

The standard paradigm for the basic mode of operation in reasoners until recently was goal-directed, backward chaining reasoning (as in Moore,1975). A simpler version of the propagation of constraint method was 'discovered' in vision research and known as the Waltz algorithm (Waltz,1972). It is the contribution of Sussman to see the potential for reasoning itself. Since then propagation of constraints has been applied to a variety of domains, especially electronic

circuit analysis and synthesis (cf. Sussman and Stallman (1976), deKleer and Sussman (1978)) but also geometry (Doyle, 1976), or common sense physics (Forbus, 1979).
A particularly interesting constraint system was developed by A. Borning(1979) and implemented in Smalltalk (Kay,1976).
A mathematical theory underlying the propagation of constraint methodology is discussed in Freuder (1976).

One important difference with the present system and other constraint systems (like the one developed by Sussman and Steele (1978)) is that these systems assume that it is possible to give a unique description (usually a numerical value) as answer to a certain question. We did not make that assumption so that we have in fact a more general system. At the same time most systems (especially Sussman and Steele (1978) and Borning (1977)) use procedural attachment as the prime method for generating new constraints rather than reasoning processes.

2. THE FRAMEWORK

There are two issues that need to be resolved in the present section. First of all we have to specify what kind of a behavior is going to occur. Second we have to show how this behavior can be performed by the computational constructs introduced in the previous chapter.

Recall that there are two components in the reasoner: one component responsible for knowledge about the domain (this is sometimes called the knowledge-base) and one component responsible for accumulating information about a particular problem situation (this was called the model).

Recall also that we proposed the knowledge-base to be a collection of frames. Each of these frames contains all the information about a certain subject-matter. Obviously these frames will have to be consulted during the construction of the model. Within the framework of the experts-architecture proposed in the previous chapter, we consequently propose to encapsulate every frame in an expert. Such an expert will be called a *frame-keeper*. Thus there will be an expert for the MOTHER-frame, the PERSON-frame, etc.
A frame-keeper has a script that specifies how it should respond to messages asking for information about the frame. Here are some typical requests that could arrive at a frame-keeper:

What are your aspects?
Give me your criterial aspects.
What description is attached to that particular aspect?
etc.

The model will consists of a set of individuals and a set of relations among these individuals. For reference inside the model we will assign a particular name to each individual that is considered in the model. Typical names are TABLE-1, or JOHN-1. We will use these names in the descriptions as if they were individual-concepts. In many cases such an individual is not an individual in the sense that it is uniquely determined

and there is no other individual like it. Instead, individuals are to be viewed as *skolem-constants* or 'anonymous individuals'.
For example, consider a description like

(A PERSON)

then we do not know which particular object is meant, instead we will consider the referent of this description to be a typical representative of the class of persons. This typical representative behaves for all purposes as a real individual except that it could at any given time be said to be identical to another individual in which case we have to *merge* the two.

The principle of object-oriented model organization prescribes that an expert should be assigned to each object in the model. We will call such an expert an *object-expert*. An object-expert for a particular individual comes into existence as a copy of the prototypical object-expert. This means concretely that there will be an expert for TABLE-15, JOHN-1, etc. This expert has the same name as the individual it is thinking about. It contains all descriptions which are true for the individual in the model.

The script of an object-expert contains rules for performing the reasoning activity itself. The basic activity in the system consists in sending a description to an object-expert. This action has the force of a predication: It says that the description holds for the individual for which the object-expert is responsible. The rules in the script of the object-expert will examine this new description in order to incorporate it with the other knowledge this object-expert already has about that individual.

In order to incorporate new knowledge, the object-expert does two types of things to a description that comes in. When the description is a basic description (i.e. contains no connectives) then the object-expert will try to *instantiate* the description. By instantiation we mean that experts are to be created (or found) for each of the aspects of the frame used in the description. Once a description is instantiated all information associated with the description has to be propagated to all the experts involved in it. We call this phase *propagation of constraints*.

When the description that enters the expert is not a basic description, the object-expert will try to decompose it so that the descriptions involved become basic descriptions. Decomposition is achieved by applying rules of inference like AND-elimination, DOUBLE-NEGATION elimination, etc.

We will now study these processes in somewhat more detail.

2. 1. INSTANTIATION

The process of instantiation involves two steps. First the reasoner must try to recover the instantiation referred to by the description, if it exists already. This phase, which we will call the *search phase*, is necessary because we want to be able to deal with *incremental descriptions*. If the instantiation could not be found, new experts have to be

created for each of the aspects in the frame involved that do not yet have known fillers. We call this phase the *creation phase.*

First we look at the search phase. Let us introduce an example to make clear what the task of this phase consists of. Given a description for MARY-1 as follows:

```
(THE MOTHER OF A FAMILY)
```

then because no instantiation already exists, the reasoner will create a new one, which could for example look like this:

```
(FAMILY
    (WITH MOTHER MARY-1)
    (WITH SELF FAMILY-1)
    (WITH FATHER FATHER-1)
    (WITH CHILD CHILD-1))
```

Now next time the description

```
(THE MOTHER OF A FAMILY)
```

enters the expert for MARY-1, no new instantiation should be created (because a person can only be the mother of one family). Instead in the search phase, it should be recovered that

```
(FAMILY
    (WITH MOTHER MARY-1)
    (WITH SELF FAMILY-1)
    (WITH FATHER FATHER-1)
    (WITH CHILD CHILD-1))
```

is the instantiation behind this new description. How should the investigation to find this instantiation proceed? We will consider two cases: for individual descriptions and for non-individual descriptions.

(i) FINDING THE INSTANCE OF AN INDIVIDUAL DESCRIPTION

First we consider the problem of finding the instance of an individual description, i.e. a description which makes use of an individual-concept. Recall that an individual description is a description whose view is individuating. For example if we have a frame for John_Doe as follows

```
(JOHN_DOE
  (WITH SELF)
  (WITH AGE)
  (ASPECT-SPECIFICATIONS:
   (INDIVIDUATING: SELF)))
```

then we know that a description like

```
JOHN_DOE
```

or

```
(THE AGE OF JOE)
```

is individuating.

In order to deal with individual descriptions, the reasoner must keep track of the descriptions which have appeared already and of the individuals which have been created to deal with them. Thus the method for finding out whether an individual concept has already been instantiated is to check whether there is already a specific referent for

the individuating aspect of this concept. If so, the reasoner can retrieve the filler. Once the filler of the individuating slot is found the reasoner can go to that expert and retrieve the rest of the instantiation.

(ii) FINDING THE INSTANCE OF A NON-INDIVIDUAL DESCRIPTION

In order to find the instance of a description which is not an individual description each object-expert goes through the following algorithm.

1. Select a series of criterial aspects, because if a referent can be found for each of these aspects and if there is an instantiated description in one of these experts making use of the concept in the description with the same view, then the reasoner can extract all information from this instantiated description.
2. Try to find the referents for each of the aspects in this series, based on a description attached to the aspect in the original description or a description attached to the aspect in the frame for the concept in the original description.
3. If all referents are found, try to find an instantiated description in one of the referents that contains the concept of the original description, the same view as the target-referent plays in the original description, and the same fillers for the other aspects in the series.
4. If this description is found, the instantiation has been discovered.
5. Further complications arise if one of the aspects is non-projective. In that case, try to find the referent of this non-projective aspect based on the description attached to the frame or to the description. If this referent cannot be found, the instantiation-group has been found but not the particular instantiation.

If any of these steps fails, it is concluded that no instantiation already exists in the model for this description.

An example will make clear what is going on here. Suppose we have a frame for a line-segment:

```
(LINE-SEGMENT
  (WITH SELF)
  (WITH BEGIN)
  (WITH END)
  (WITH DISTANCE)
  (ASPECT-SPECIFICATIONS:
   (CRITERIAL: (BEGIN END) (SELF))))
```

and two individuals POINT-1 and POINT-2 which are described as A and B. Suppose POINT-1 also contains the description

```
(THE BEGIN OF A LINE-SEGMENT
        (WHICH IS LINE-1)
        (WITH END POINT-2)
        (WITH DISTANCE DISTANCE-1))
```

We will consider the problem of finding the instance of the following description:

```
(A LINE
   (WITH BEGIN A)
   (WITH END B))
```

The result should be

```
(LINE-SEGMENT
  (WITH SELF LINE-1)
  (WITH BEGIN POINT-1)
  (WITH END POINT-2)
  (WITH DISTANCE DISTANCE-1))
```

Let us go through the algorithm. First we have to pick a series of criterial aspects. Let us pick the first one, namely (BEGIN END). This means that the begin and end of the line-segment frame are criterial. According to step 2 we have to find the referents of the descriptions attached to these aspects. This causes a recursive call to the method for finding the referent of a description. The description attached to the begin-aspect is A and to the end-aspect is B. A and B are both individual-descriptions, we will assume that their referents can be found and that the result is POINT-1 and POINT-2 respectively.
The next step is to see whether any of these experts contains a description making use of the concept of LINE-SEGMENT, such that the view is the same and the aspects besides the view are filled with the same individuals as predicted. In other words we check whether POINT-1 has a description which matches with the following source-description

```
(THE BEGIN OF A LINE-SEGMENT
     (WITH END POINT-2))
```

By matching we mean that all aspects in the source-description have the same filler as in the target-description but there could be more aspects in the target-description.

Now we see that POINT-1 contains the following description:

```
(THE BEGIN OF A LINE-SEGMENT
          (WHICH IS LINE-1)
          (WITH END POINT-2)
          (WITH DISTANCE DISTANCE-1)).
```

which indeed matches with

```
(THE BEGIN OF A LINE-SEGMENT
     (WITH END POINT-2))
```

From this we can extract the instance which is

```
(LINE-SEGMENT
 (WITH SELF LINE-1)
 (WITH BEGIN POINT-1)
 (WITH END POINT-2)
 (WITH DISTANCE DISTANCE-1))
```

(iii) FINDING THE REFERENT OF A DESCRIPTION

The algorithm for discovering the instance of a description, involved a step were the reasoner must find the referent of a description. The referent of a description can easily be deduced once it is known what instance is referred to by the description. In particular, the referent is the filler of the view of the description in that particular instance. For example, we can find the referent of the following description

```
(A LINE-SEGMENT
   (WITH BEGIN A)
   (WITH END B))
```

easily, once we know that the instance referred to by this description is

```
(LINE-SEGMENT
 (WITH SELF LINE-1)
 (WITH BEGIN POINT-1)
 (WITH END POINT-2)
 (WITH DISTANCE DISTANCE-1))
```

The referent is LINE-1, because LINE-1 fills the self-slot of this instance.

Notice however that the referent of the description is only equal to the filler of the view if the view is a projective aspect. If it is non-projective, we cannot know what the referent is.

(iv) COMPLICATIONS DUE TO CO-REFERENTIAL LINKS.

When there are co-referential links in a frame or in the description to be instantiated, things become more complicated. As soon as the filler of one slot has been found, the results are to be propagated to the other descriptions attached in the frame. The reasoner often has to go through several cycles of comparing descriptions and trying to accumulate constraints on the co-referential links until the instantiation can be uniquely determined.

Here is an example. Supose we have a frame for family which looks like this

```
(FAMILY
 (WITH SELF)
 (WITH FATHER (A HUSBAND
                 (WITH WIFE (= THE-MOTHER))))
 (WITH MOTHER)
 (ASPECT-SPECIFICATIONS:
  (CRITERIAL: (SELF) (FATHER) (MOTHER)))).
```

Also we know that John-1 is described as

```
(A HUSBAND (WITH WIFE MARY-1))
```

making reference to a frame for husband which look like this:

```
(HUSBAND
 (WITH SELF)
 (WITH WIFE)
 (ASPECT-SPECIFICATIONS:
  (CRITERIAL: (WIFE) (SELF))))
```

and that Mary-1 is described as

```
(THE MOTHER OF A FAMILY
      (WITH SELF FAMILY-1)
      (WITH FATHER JONES-1)).
```

The problem is to find the instantiation of the following description for John-1:

```
(THE FATHER OF A FAMILY)
```

Again we go through the algorithm. First we try to find the referent for the father-aspect. John-1 is not yet described as the father. On the other hand the description attached to the father-aspect in the family-frame is

```
(A HUSBAND (WITH WIFE (= THE-MOTHER)))
```

This yields a referent because John-1 is described as

```
(A HUSBAND (WITH WIFE MARY-1))
```

and a person can only once be the self of a husband. Based on this match the referent of the co-referential description (= THE-MOTHER) in the family-frame becomes known. We had no luck in finding the family-instance based on the father. But mother is also a criterial aspect so we investigate that angle. The filler of this aspect is MARY-1. So a request is sent to that expert asking whether it contains a description using the concept of a family. Indeed it does: Mary-1 is described as

```
(THE MOTHER OF A FAMILY
    (WITH SELF FAMILY-1)
    (WITH FATHER JONES-1))
```

from this we can extract the other fillers. And the resulting instantiation is

```
(FAMILY
 (WITH MOTHER MARY-1)
 (WITH SELF FAMILY-1)
 (WITH FATHER JONES-1))
```

But now notice a certain anomaly. The filler of the father-aspect in this description is JONES-1, but we knew already that the filler of the father-aspect in this particular instantiation of the family-frame is JOHN-1. There is only one conclusion : JONES-1 and JOHN-1 are co-referential. What we have to do in such a case is establish an *identity-link* between the two object-experts which makes them virtualy identical. We cannot make them actually identical because the identity might be context-dependent. (More on contexts will be said later on.)

This is a typical example of the kind of searching and relaxation that goes on when the reasoner tries to find the instantiation of a description.

In many cases there is not yet an instantiation for the description that enters a certain object-expert. In that case a new instantiation has to be created. This means that new experts have to be created for each of the aspects that do not have a filler yet. For example, the first time the following description enters the expert for MARY-1

```
(THE MOTHER OF A FAMILY)
```

a new instantiation will be created:

```
(FAMILY
  (WITH SELF FAMILY-1)
  (WITH MOTHER MARY-1)
  (WITH FATHER FATHER-1))
```

2. 2. PROPAGATION OF CONSTRAINTS

Given a certain description and its instantiation the object-expert can proceed to propagate all the information that is associated with the new description. By propagation we mean that the expert updates its own database appropriately and that it tells the other experts involved in this instantiation about this description. Information to be propagated comes from three sources:

(i) An expert is always told what its role is in a certain instantiation. For example, if we have an instantiation like

```
(HUSBAND
  (WITH SELF JOHN-1)
  (WITH WIFE MARY-1))
```

then JOHN-1 has a description of the form

```
(A HUSBAND (WITH WIFE MARY-1))
```

and MARY-1 will have a description of the form

```
(THE WIFE OF A HUSBAND (WHO IS JOHN-1))
```

(ii) The expert of each aspect should also be given the description that is attached to the frame (with co-referential descriptions resolved by their referents in this particular instantiation).
For example, if we have the following frame

```
(MOTHER
  (WITH SELF
        (A PARENT (WITH CHILD (= THE-CHILD))))
  (WITH CHILD))
```

then given an instantiation like

```
(MOTHER
  (WITH SELF MARY-1)
  (WITH CHILD GEORGE-1))
```

MARY-1 will be sent a description of the form

```
(A PARENT (WITH CHILD GEORGE-1))
```

(iii) Finally we have to distribute the descriptions that were attached to the description causing the instantiation. For example, if we have the following description

```
(THE CHILD OF A MOTHER
   (WHO IS
         (THE WIFE OF A HUSBAND (WHO IS GEORGE-1))))
```

for MARY-1, and if we have been able to recover that the instantiation underlying this description is

```
(MOTHER
 (WITH SELF MARY-1)
 (WITH CHILD GEORGE-1))
```

then MARY-1 should be sent the description

```
(THE WIFE OF A HUSBAND (WHO IS GEORGE-1))
```

Each of the descriptions which thus arives at an expert will again be instantiated causing other descriptions to be sent around, etc. At each moment of time each of the object-experts in the model might be looking at new descriptions and trying to find out the instantiation and its consequents. We get a high form of parallellism in the reasoner because activities in one object-expert are not at all disturbed by activities in other experts.

One final technical point. There is a clear distinction to be made between descriptions which are not yet instantiated (and which therefore need full attention by the reasoner) nd descriptions which are already instantiated. For example if a certain instantiation is

propagated to a series of object-experts, we do not want each of these messages telling about this instantiation to cause another attempt to find the involved instantiation. In order to make the difference between the two states of a description clear, we introduce *instantiation-markers* in front of a description. The instantiation-marker for uninstantiated is YOU-ARE-DESCRIBED-AS and the instantiation-marker for instantiated is IS. Thus the following description:

```
(you-are-described-as (THE MOTHER OF A FAMILY))
```

is an uninstantiated description, whereas the following one

```
(is (THE MOTHER OF A FAMILY
        (WHICH IS FAMILY-1)
        (WITH FATHER FATHER-1))
```

might be its instantiated counterpart.

2. 3. DECOMPOSITION

When a description contains connectives it cannot be instantiated right away. Instead the description has to be decomposed until basic descriptions result. In order to do this we can use well known rules of inference from propositional calculus. First we deal with the connectives, then with conditional descriptions.

(I) THE CONNECTIVES

A typical decomposition rule is AND-elimination, which says that given a conjunction of descriptions, predicate each of the descriptions. For example, if we have a description of the form

```
(AND (THE MOTHER OF A FAMILY)
     (THE WIFE OF A HUSBAND))
```

this description can be decomposed into two seperate ones:

```
(THE MOTHER OF A FAMILY)
```

and

```
(THE WIFE OF A HUSBAND)
```

Because each of these descriptions is now a basic description, it can be instantiated based on the methods we discussed in previous paragraphs.

How would we mechanize such a rule given the computational apparatus proposed in the previous chapter. We can do this using the pattern-directed invocation rule idea. For each inference-rule, we can have a CONTINUOUS-ASK that will look out for descriptions which match the condition of the inference rule. The body of the rule contains then further message-passing as specified by the inference-rule. For the case of AND-elimination we might have a rule which looks like this:

```
(CONTINUOUS-ASK :MYSELF
                (you-are-described-as (AND :DESCRIPTION-1 :DESCRIPTION-2))
                (TELL :MYSELF
                      (you-are-described-as :DESCRIPTION-1))
                (TELL :MYSELF
                      (you-are-described-as :DESCRIPTION-2)))
```

This rule will be put in the prototypical object-expert and will therefore be inherited by each object-expert that ever gets created. The rule looks out for incoming messages of the form

```
(you-are-described-as (AND :DESCRIPTON-1 :DESCRIPTION-2))
```

For example, suppose the following description enters MARY-1:

```
(you-are-described-as
 (AND (THE MOTHER OF A FAMILY)
      (THE WIFE OF A HUSBAND)))
```

then this description would match with the condition in the rule where :DESCRIPTION-1 is bound to

```
(THE MOTHER OF A FAMILY)
```

and :DESCRIPTION-2 to

```
(THE WIFE OF A HUSBAND)
```

The body of the rule can now be executed which yields two new message-passing activities:

```
(TELL MARY-1
      (you-are-described-as (THE MOTHER OF A FAMILY)))
```

and

```
(TELL MARY-1
      (you-are-described-as (THE WIFE OF A HUSBAND)))
```

Here are some other examples of inference-rules that have been implemented in a similar fashion.

An exclusive disjunction of description can be eliminated if the negation of the other disjuncts. For example, the connective in the following description

```
(XOR (A MALE-PERSON) (A FEMALE-PERSON))
```

can be eliminated in the following ways. When it becomes known that

```
(NOT (A MALE-PERSON))
```

we can deduce that

```
(A FEMALE-PERSON)
```

and when it becomes known that

```
(A MALE-PERSON)
```

we can deduce that

```
(NOT (A FEMALE-PERSON))
```

Similarly for the second description.

A non-exclusive disjunction can only be eliminated if the negation of all other disjuncts is known. The two negations of a double-negation can be eliminated, etc.

Rather than discuss other well-known inference rules, we will turn our attention to negation of a basic desciption like

```
(NOT (A FATHER))
```

Notice first of all that negations cannot be instantiated in the same way as positive descriptions for the simple reason that the properties that are attached in the frame do not necessarily hold for the negation, and neither does the negation of these properties

hold. For example if we have a frame for WATER:

```
(WATER
 (WITH SELF (A LIQUID)))
```

and we now say

```
(NOT WATER)
```

then we do not want do conclude (NOT (A LIQUID)) because the object in question might very well be a different liquid than water.

On the other hand, it is not possible to leave the negated description uninstantiated because if the matcher wants to find out whether a certain description matches with another one it can only do so if the slot-fillers are described by their individual names, and this instantiated status can only be achieved by instantiating the description. For example, suppose we have

```
(NOT (THE FATHER OF A FAMILY
          (WITH MOTHER (THE WIFE OF A HUSBAND (WHO IS GEORGE)))))
```

then we need to know the individual-name of the mother within the current model in order to use this negative description.

These things are puzzling but fortunately we found an interesting solution: Whenever we encounter a negation, we instantiate the positive description with a new anonymous individual as the view of the description. Then we tell the slot-filler of the slot to which the description is attached that it is not equal to this new anonymous individual and that it does not play the role of the view in the (now instantiated) description.

This method works because the expert created for the anonymous individual starts developing the consequents of having that description and if it is ever discovered that the anonymous individual is identical to this new anonymous invidiual, a conflict will occur, i.e. the system realizes it has arrived at a contradiction. At the same time we can use the negation in matching because the actual object is described by the instantiated negative description.

The introduction of negations introduces the problem of maintaining consistency in the model. A model is inconsistent if both a description and its negation have been predicated for the same object. When this fact is noticed by an object-expert it will emit a conflict message. Examples of this will be studied later.

(ii) CONDITIONAL DESCRIPTIONS

Conditional descriptions like

```
(IF THE-PARENT IS
    ((A MALE-PERSON)
     (A FATHER (WITH CHILD (= THE-CHILD)))))
```

are decomposed using the rule of Modus Ponens. The question is how this can be mechanized given the computational apparatus introduced before.

Again we can set up pattern-directed invocation rules for each of these descriptions. In general the condition of such a rule will correspond with the condition in the conditional

description, the type of rule will correspond with the condition-indicator and the action if there is a match will consist in sending a description to the appropriate expert predicating the resulting description.

Here are some more details. Let us look at the first type of condition- indicator. In particular IF-NOW (an instantaneous/sequential conditional). This will lead to a rule based on an INSTANTANEOUS-ASK. Because the conditional is sequential, we make an embedded rule for each condition in the description.

Consider the following example:

```
(IF-NOW MARY-1 IS
     ((A MALE-PERSON)
      (A FATHER (WITH CHILD GEORGE-1)))
     ((A FEMALE-PERSON
         (A MOTHER (WITH CHILD GEORGE-1))))
     (ELSE
         (A PARENT (WITH CHILD GEORGE-1))))
```

Suppose this description enters the expert MARY-1. Then this will cause the following rule to be set up:

```
(INSTANTANEOUS-ASK MARY-1
      (is (A MALE-PERSON))
      ;; action if match:
      (TELL MARY-1
            (you-are-described-as (A FATHER (WITH CHILD GEORGE-1)))
      ;; action if failure:
      (TELL MARY-1
            (you-are-described-as
               (IF MARY-1 IS
                  ((A FEMALE-PERSON
                      (A MOTHER (WITH CHILD GEORGE-1))))
                  (ELSE
                   (A PARENT (WITH CHILD GEORGE-1)))))))
```

Because MARY-1 is not described as a male-person the action if failure is executed which will cause the creation of a new rule:

```
(INSTANTANEOUS-ASK MARY-1
      (is (A FEMALE-PERSON))
      ;; action if match:
      (TELL MARY-1
            (you-are-described-as
             (A MOTHER (WITH CHILD GEORGE-1))))
      ;; action if failure:
      (TELL MARY-1
            (you-are-described-as
                 (A PARENT (WITH CHILD GEORGE-1)))))
```

This will not match either and the action if failure will be executed.

There are of course all sorts of technical details associated with mechanizing this process. For example bindings for co-referential descriptions could be discovered which then have to be added to the environment of the closures created for each of the TELL actions, etc. But the main line of thought should be clear from these examples.

Other conditionals are built up using the other rule-types. For example a SINGLE-EVENT/PARALLEL conditional can be set up using the SINGLE-EVENT-ASK rule-type. Rather than create embeddings for the various conditions, we create a seperate rule for each conditional. Thus the following description for MARY-1

```
(IF MARY-1 IS
    ((A FEMALE-PERSON) (A MOTHER (WITH CHILD GEORGE-1)))
    ((A MALE-PERSON) (A FATHER (WITH CHILD GEORGE-1))))
```

will lead to the following two rules in the script of the expert for MARY-1:

```
(SINGLE-EVENT-ASK MARY-1
    (IS (A FEMALE-PERSON))
    (TELL MARY-1
          (you-are-described-as (A MOTHER (WITH CHILD GEORGE-1)))))

(SINGLE-EVENT-ASK MARY-1
    (IS (A MALE-PERSON))
    (TELL MARY-1
          (you-are-described-as (A PARENT (WITH CHILD GEORGE-1)))))
```

Finally if you have a CONTINUOUS/PARALLEL conditional, like

```
(WHEN MARY-1 IS
      ((A FEMALE-PERSON) (A MOTHER (WITH CHILD GEORGE-1)))
      ((A MALE-PERSON) (A FATHER (WITH CHILD GEORGE-1))))
```

then the following rules will be created

```
(CONTINOUS-ASK MARY-1
    (IS (A FEMALE-PERSON))
    (TELL MARY-1
          (you-are-described-as (A MOTHER (WITH CHILD GEORGE-1)))))

(CONTINOUS-ASK MARY-1
    (IS (A MALE-PERSON))
    (TELL MARY-1
          (you-are-described-as (A PARENT (WITH CHILD GEORGE-1)))))
```

The condition in a conditional must always be a basic-description with no descriptions attached to its slots, except names of individuals in the model or co-referential descriptions that are to bound by the match.

Further complications therefore arise if the condition in a conditional is itself a complex description with connectives. When the condition is a conjunction of conditions, we set up a pattern-directed invocation rule in the expert reasoning about the referring-name that will look out for the first conjunct. When this conjunct is found we set up a new rule for the second conjunct, etc. If all conjuncts match, the resulting description is sent to the expert reasoning about the slot-filler.
When the condition is a disjunction of conditions, we set up a pattern-directed invocation rule for each disjunct. As soon as one of them triggers, the resulting description is propagated.
When the condition is an exclusive disjunction, the triggers of the rules will contain a disjunct and the conjunction of the negation of the other disjuncts. When such a trigger is satisfied the resulting description is propagated.
Finally if descriptions are attached to a condition, we re-arrange the description so that

a new conditional is created for this further constraint. For example, the following description

```
(WHEN A-PERSON IS
      ((AND (A PARENT
                (WITH CHILD JOHN))
            (NOT (A MALE-PERSON)))
       (A MOTHER (WITH CHILD JOHN))))
```

is actually transformed first into

```
(WHEN A-PERSON IS
      ((AND (A PARENT
                (WITH CHILD (= A-DUMMY))
            (NOT (A MALE-PERSON))))
       (WHEN A-DUMMY IS
             (JOHN (A MOTHER (WITH CHILD JOHN))))))
```

What we did here was create new referring-names for each slot that contains further descriptions (or names are used if they exist already) and a new conditional is formed with this referring-name as referring-name and with the additional description as condition.

(iii) EXPLICIT PREDICATION

A finale note on how explicit predications like

```
(JOHN-1 IS (THE MOTHER OF A FAMILY))
```

are decomposed. This is easy. We have a pattern-directed invocation rule in each object-expert that looks out for descriptions of this form and then sends the second description to the expert in question.

These explanations should provide the reader with a fairly concrete picture of how reasoning behavior for the language presented in chapter 2 can be implemented given the computational primitives presented in chapter 3. In the next chapter we will give a series of concrete examples.

DISCUSSION

The reasoning processes prestend here are an exponent of existing work on reasoning. See e.g. the overview in Nilson (1973) for earlier work on reasoning and papers by Hewitt (1975), Fikes and Hendrix (1977), DeKleer, et.al.(1977) a.o. for a more recent discussion of some of the methods we proposed here. On the other hand we believe that the present system is the first operational reasoning system that is completely based on descriptions (rather than statements). This required us to find new methods for certain problems, such as negative descriptions.

The question could be raised of what is the logical validity of the reasoning schemes proposed here. We see that so far these mechanisms relate naturally to mechanisms advanced for reasoning in predicate calculus. In particular, we see that we introduced the ability to match descriptions based on the unification principle, i.e. that two descriptions match if the substitutions of the descriptions in the slots match. We can do this very effectively because we transform all descriptions into descriptions with unique names for the individuals. This was done by instantiating, which corresponds to a skolemization in predicate-calculus based theorem provers. Skolemization raises

the problem of identity that is later discovered and this has been effectively solved here by merging the experts dealing with the objects. It can be shown that the methods of unification and skolemization lead to logical completeness (Chang and Lee, 1973). For the composition of more complex descriptions, we follow basically the methods of natural deduction (cf. Kalish and Montague(1972) and Prawitz (1965).

EXAMPLES

5. EXAMPLES

Chapter 2 introduced a frame-based description language. Chapter 3 introduced the architecture and procedural primitives of a reasoning system. Chapter 4 combined proposals of chapter 2 and 3 and contained specifications for a concrete reasoning system supporting the description language of chapter 2.

This chapter contains some nontrivial examples illustrating the workings of this system. We had three goals in mind when working out these examples:

(i) They should show that the proposals have lead to working programs (all the examples have been extracted from interactions. with an existing implementation).

(ii) They should enable the reader to find out more details and develop a clearer picture of the description language and the reasoning mechanisms.
and

(iii) They should illustrate particular sorts of reasoning behavior.

The examples we will look at in this chapter come from the domain of common-sense reasoning. Other applications will be studied later.
The first example illustrates in detail the message passing that goes on by way of a demon which deals with a typical common-sense fact : if it rains, you get wet.
The second series of examples illustrates other aspects of reasoning. Within the context of the generalization hierarchy for possession-transfer introduced earlier on, we will give clear examples showing how inheritance works, how instantiations are triggered going up and down hierarchies, how the merging of descriptions operates, how identity is established and how negation works.
The final example is taken from the domain of algorithmic common-sense reasoning, in particular we will deal with the infamous flush-toilet example. The example illustrates part-whole hierarchies, model construction and descriptions from multiple viewpoints. It should provide the reader with a global idea of what is possible with the mechanisms proposed so far.

1. COMMON SENSE DEMONS

A lot of common sense knowledge consists in knowing all sorts of tiny little facts that pop up when one is thinking about related objects. This activity is often compared to a demon which jumps up when certain conditions in a model are met. The following example introduces such a demon. It knows about the fact that physical objects get wet when it rains.

We will need the following concepts:
- a concept for a physical-object:

```
(PHYSICAL-OBJECT
  (WITH SELF))
```

- a concept for the state of being located-at a certain location in a certain situation:

```
(LOCATED-AT-STATE
  (WITH SELF)
  (WITH OBJECT)
  (WITH LOCATION)
  (WITH SITUATION))
```

- a concept for an outside location:

```
(OUTSIDE-LOCATION
  (WITH SELF))
```

- a concept for a wet-object in a certain situation

```
(WET-OBJECT
  (WITH SELF)
  (WITH SITUATION))
```

- a concept for a situation where it rains:

```
(RAIN-SITUATION
  (WITH SELF))
```

The information that physical objects get wet when it rains can be expressed by attaching a conditional description to the physical-object frame which looks out whether the object is located at a location which is an outside location in a situation which is a rain-situation. If that is the case the object is described as a wet object.

```
(PHYSICAL-OBJECT
      (WITH SELF (= THE-OBJECT)
            ;; when the object has a particular location
            (WHEN THE-OBJECT IS
                  ((THE OBJECT OF A LOCATED-AT-STATE
                        (WITH LOCATION (= THE-LOCATION))
                        (WITH SITUATION (= THE-SITUATION)))
                   ;; when there is rain in this situation
                   (WHEN THE-SITUATION IS
                         ((A RAIN-SITUATION)
                          ;; and when the location is an outside location
                          (WHEN THE-LOCATION
                                ((AN OUTSIDE-LOCATION)
                                 ;; then the object gets wet
                                 (THE-OBJECT IS
                                     (A WET-OBJECT
                                        (WITH SITUATION (= THE-SITUATION))))))))))))
```

Also we introduce a frame for John which is described as a physical object, located-at a certain outside-location :

```
(JOHN
  (WITH SELF
        (AND
         (A PHYSICAL-OBJECT)
         (THE OBJECT OF A LOCATED-AT-STATE
              (WITH LOCATION (AN OUTSIDE-LOCATION))))))
```

We will now try to illustrate some aspects of the reasoning process. The interaction and output is at the very lowest level of the reasoning process, i.e. the message passing level.

We start with the following message:

```
>> (tell (an OBJECT)
         (you-are-described-as JOHN))
```

The first thing what happens is that a new copy of the prototypical object-expert is created (further called JOHN-1) and the message is sent to this new expert:

```
===> JOHN-1 receives the following message from USER:
(you-are-described-as JOHN)
```

An object-expert always tells itself its own name for later identification purposes:

```
===> JOHN-1 receives the following message from JOHN-1:
(is JOHN-1)
```

JOHN-1 now starts to instantiate the description. The instantiation in this case is fairly simple because no new object-experts need to be created for other aspects. The only thing that happens is that John tells itself its role in the John concept:

```
===> JOHN-1 receives the following message from JOHN:
(is JOHN)
```

Now JOHN-1 asks the frame-keeper for JOHN whether it knows any constraints that objects which are described as the self of JOHN have. This frame-keeper replies with the description attached to the self-aspect:

```
===> JOHN-1 receives the following message from JOHN:
(you-are-described-as
    (AND
     (A PHYSICAL-OBJECT)
     (THE OBJECT OF A LOCATED-AT-STATE
          (WITH LOCATION (AN OUTSIDE-LOCATION)))))
```

JOHN-1 looks at this message and uses its AND-elimination rule to decompose the expression and to send each of the conjuncts as new descriptions to itself:

```
===> JOHN-1 receives the following message from JOHN-1:
(you-are-described-as
   (A PHYSICAL-OBJECT))
===> JOHN-1 receives the following message from JOHN-1:
(you-are-described-as
   (THE OBJECT OF A LOCATED-AT-STATE
        (WITH LOCATION (AN OUTSIDE-LOCATION))))
```

JOHN-1 looks now both descriptions and will try to instantiate each of them. This leads to the following message passing behavior:

```
===> JOHN-1 receives the following message from PHYSICAL-OBJECT:
(is (A PHYSICAL-OBJECT))
```

The PHYSICAL-OBJECT frame also sends a constraint along. Note how the indirect-reference THE-OBJECT has been resolved via a binding to JOHN-1:

```
===> JOHN-1 receives the following message from PHYSICAL-OBJECT:
(you-are-described-as
        (WHEN JOHN-1 IS
              ((THE OBJECT OF A LOCATED-AT-STATE
                    (WITH LOCATION (= THE-LOCATION))
                    (WITH SITUATION (= THE-SITUATION)))
               (WHEN THE-SITUATION IS
                    ((A RAIN-SITUATION)
                     (WHEN THE-LOCATION IS
                          ((AN OUTSIDE-LOCATION)
                           (JOHN-1 IS
                                 (A WET-OBJECT
                                    (WITH SITUATION (= THE-SITUATION)))))))))))
```

Now we turn to the other description, i.e.

```
(you-are-described-as
   (THE OBJECT OF A LOCATED-AT-STATE
        (WITH LOCATION (AN OUTSIDE-LOCATION))))
```

We notice first of all that this is a partial description (the situation is not mentioned). In any event new object-experts need to be created for each of the aspects of the frame that does not have an expert reasoning about it. This new experts will be called SITUATION-1 (for the situation aspect) LOCATED-AT-STATE-1 (for the self aspect) and LOCATION-1 (for the location aspect). Whenever a new expert is created it first tells itself its name, so we get

```
===> SITUATION-1 receives the following message from SITUATION-1
(is SITUATION-1))
===> LOCATED-AT-STATE-1 receives the following message from LOCATED-AT-STATE-1
(is LOCATED-AT-STATE-1)
===> LOCATION-1 receives the following message from LOCATION-1
(is LOCATION-1)
```

These new experts also receive descriptions specifying their role in the frame

```
===>  LOCATION-1 receives the following message from LOCATED-AT-STATE:
(is (THE LOCATION OF A LOCATED-AT-STATE
         (WITH SELF LOCATED-AT-STATE-1)
         (WITH SITUATION SITUATION-1)
         (WITH OBJECT JOHN-1)))
===>  LOCATED-AT-STATE-1 receives the following message from LOCATED-AT-STATE:
(is (A LOCATED-AT-STATE
       (WITH LOCATION LOCATION-1)
       (WITH SITUATION SITUATION-1)
       (WITH OBJECT JOHN-1)))
===>  SITUATION-1 receives the following message from LOCATED-AT-STATE:
(is (THE SITUATION OF A LOCATED-AT-STATE
         (WITH SELF LOCATED-AT-STATE-1)
         (WITH LOCATION LOCATION-1)
         (WITH OBJECT JOHN-1)))
===>  JOHN-1 receives the following message from LOCATED-AT-STATE:
(is (THE OBJECT OF A LOCATED-AT-STATE
         (WITH SELF LOCATED-AT-STATE-1)
         (WITH SITUATION SITUATION-1)
         (WITH LOCATION LOCATION-1)))
```

They also receive the descriptions that were attached to the description which was the

source for this particular instantiation:

```
===> LOCATION-1 receives the following message from JOHN-1:
(you-are-described-as (AN OUTSIDE-LOCATION))
```

This description is again further instantiated leading to

```
===> LOCATION-1 receives the following message from OUTSIDE-LOCATION:
(is (AN OUTSIDE-LOCATION))
```

Meanwhile the condition in the conditional-description waiting in JOHN-1 matches with the description that JOHN-1 is the object of a located-at-situation. Based on this match, the resulting description is released:

```
===> JOHN-1 receives the following message from JOHN-1
(you-are-described-as
      (WHEN SITUATION-1 IS
            ((A RAIN-SITUATION)
             (WHEN LOCATION-1 IS
                   ((AN OUTSIDE-LOCATION)
                    (JOHN-1 IS
                       (A WET-OBJECT
                          (WITH SITUATION SITUATION-1)))))))))
```

Note how the indirect references have been resolved based on the matching process.

The model contains now everything that can be known at this point. A condition is waiting for SITUATION-1 to be described as (A RAIN-SITUATION). So let us send such a description to SITUATION-1:

```
>> (tell SITUATION-1 (you-are-described-as (A RAIN-SITUATION)))
===> SITUATION-1 receives the following message from USER:
(you-are-described-as (A RAIN-SITUATION))
```

After instantiation this leads to

```
===> SITUATION-1 receives the following message from RAIN:
.(is (A RAIN-SITUATION))
```

The instantiation of this description matches with the condition. So a new description results:

```
===> JOHN-1 receives the following message from SITUATION-1:
(you-are-described-as
    (WHEN LOCATION-1 IS
          ((AN OUTSIDE-LOCATION)
           (JOHN-1 IS
                (A WET-OBJECT
                   (WITH SITUATION SITUATION-1)))))
```

But LOCATION-1 was already described as (AN OUTSIDE-LOCATION), hence the condition matches and JOHN-1 is described as being wet:

```
===> JOHN-1 receives the following message from LOCATION-1:
(you-are-described-as (A WET-OBJECT (WITH SITUATION SITUATION-1)))
```

Instantiation of this description leads to

```
===> JOHN-1 receives the following message from WET-OBJECT:
(is (A WET-OBJECT (WITH SITUATION SITUATION-1)))
===> SITUATION-1 receives the following message from WET-OBJECT:
(is (THE SITUATION OF A WET-OBJECT (WITH SELF JOHN-1)))
```

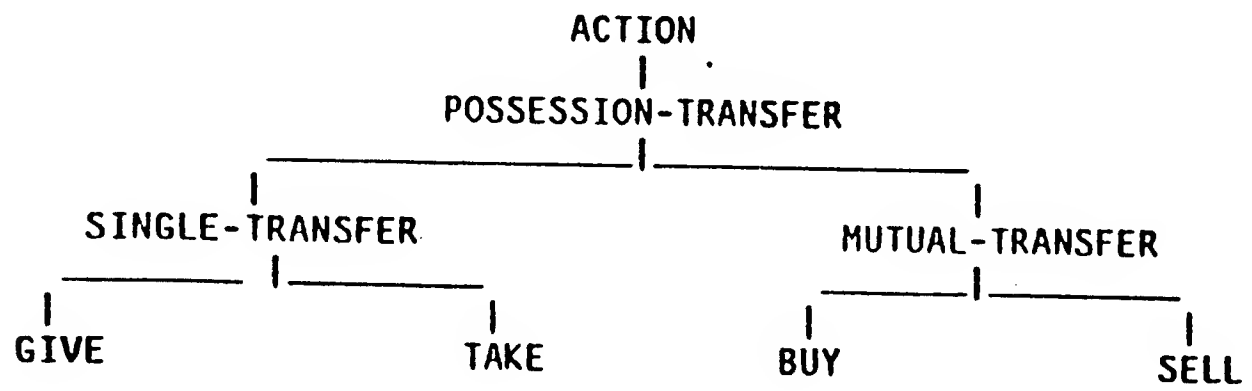
Looking at the messages that arrive at the experts does not give a good overview of the

process or the results that have been achieved. Fortunately there are two views one can maintain on the process of model-construction: one can look at it from the viewpoint of the objects (i.e. see what messages come in - as we did here) or one can look at it from the viewpoint of the frames (i.e. see what instantiations are being made). The second viewpoint turns out to yield a better overview of the process. The following is a list of the instantiations for the previous example:

```
>> (tell (an object) (you-are-described-as JOHN))
```
causes the construction of the following instantiations
```
**
(JOHN (WITH SELF JOHN-1))
**
(PHYSICAL-OBJECT (WITH SELF JOHN-1))
**
(LOCATED-AT-STATE (WITH SELF LOCATED-AT-STATE-1)
                  (WITH LOCATION LOCATION-1)
                  (WITH OBJECT JOHN-1)
                  (WITH SITUATION SITUATION-1))
**
(OUTSIDE-LOCATION (WITH SELF LOCATION-1))
```
When we now tell the relevant expert in the model that it rains:
```
>> (tell SITUATION-1 (you-are-described-as (A RAIN-SITUATION)))
```
the following additional instantiations are created:
```
**
(RAIN-SITUATION (WITH SELF SITUATION-1))
**
(WET-OBJECT (WITH SELF JOHN-1)
            (WITH SITUATION SITUATION-1))
```

2. HIERARCHIES

The next series of examples is set up to illustrate in detail various technical points. All the examples are based on the generalization hierarchy for possession-transfer discussed in chapter 2:



We will use the frames that were given in that chapter, in addition to some frames for JOHN, MARY and BOOK which have no constraints attached.

2. 1. UPWARD MOVEMENT IN HIERARCHIES

First we show how one can go from one node in the hierarchy to a node above it, in particular from the frame for BUY via instantiations of mutual-transfer and possession-transfer to action.

The following message is sent to the prototypical object-expert. Observe that the aspects about which nothing can be said are simply left out, in other words, this is a partial description. Descriptions attached to the old-owner, new-owner and object will have to be sent to the object experts created due to instantiation of buy.

```
>> (tell (an OBJECT)
        (you-are-described-as
              (A BUY
                 (WITH OLD-OWNER JOHN)
                 (WITH NEW-OWNER MARY)
                 (WITH OBJECT (A BOOK)))))
```

Results in the following instantiations:

```
**
(BUY (WITH SELF BUY-1)
     (WITH OBJECT OBJECT-1)
     (WITH OLD-OWNER JOHN-1)
     (WITH NEW-OWNER MARY-1)
     (WITH EXCHANGE-OBJECT EXCHANGE-OBJECT-1)
     (WITH BEGIN-SITUATION BEGIN-SITUATION-1)
     (WITH END-SITUATION END-SITUATION-1))
**
(MUTUAL-TRANSFER (WITH SELF BUY-1)
                 (WITH ACTOR MARY-1)
                 (WITH OLD-OWNER OLD-OWNER-1)
                 (WITH NEW-OWNER MARY-1)
                 (WITH OBJECT OBJECT-1)
                 (WITH EXCHANGE-OBJECT EXCHANGE-OBJECT-1)
                 (WITH BEGIN-SITUATION BEGIN-SITUATION-1)
                 (WITH END-SITUATION END-SITUATION-1))
**
(UNORDERED-COMPOSITION-OF-ACTIONS (WITH SELF BUY-1)
                                  (WITH ONE-ACTION ONE-ACTION-1)
                                  (WITH OTHER-ACTION OTHER-ACTION-1))
**
(BOOK (WITH SELF OBJECT-1))
**
(JOHN (WITH SELF JOHN-1))
**
(MARY (WITH SELF MARY-1))
**
(AMOUNT-OF-MONEY (WITH SELF EXCHANGE-OBJECT-1))
**
(POSSESSION-TRANSFER (WITH SELF ONE-ACTION-1)
                     (WITH ACTOR MARY-1)
                     (WITH OLD-OWNER OLD-OWNER-1)
                     (WITH NEW-OWNER MARY-1)
                     (WITH OBJECT OBJECT-1)
                     (WITH BEGIN-SITUATION BEGIN-SITUATION-1)
```

```
                        (WITH END-SITUATION END-SITUATION-1))
**
(POSSESSION-TRANSFER (WITH SELF OTHER-ACTION-1)
                     (WITH ACTOR MARY-1)
                     (WITH OLD-OWNER MARY-1)
                     (WITH NEW-OWNER OLD-OWNER-1)
                     (WITH OBJECT EXCHANGE-OBJECT-1)
                     (WITH BEGIN-SITUATION BEGIN-SITUATION-1)
                     (WITH END-SITUATION END-SITUATION-1))
**
(ACTION (WITH SELF BUY-1)
        (WITH ACTOR MARY-1)
        (WITH BEGIN-SITUATION BEGIN-SITUATION-1)
        (WITH END-SITUATION END-SITUATION-1))
**
(POSSESSION (WITH SELF POSSESSION-1)
            (WITH HAVER JOHN-1)
            (WITH OBJECT OBJECT-1)
            (WITH SITUATION BEGIN-SITUATION-1))
**
(POSSESSION (WITH SELF POSSESSION-2)
            (WITH HAVER MARY-1)
            (WITH OBJECT OBJECT-1)
            (WITH SITUATION END-SITUATION-1))
**
(ACTION (WITH ACTION (CALLED XPRT-23))
        (WITH ACTOR MARY-1)
        (WITH BEGIN-SITUATION BEGIN-SITUATION-1)
        (WITH END-SITUATION END-SITUATION-1))
**
(TIME-SEQUENCE (WITH SELF TIME-SEQUENCE-1)
               (WITH FIRST-SITUATION BEGIN-SITUATION-1)
               (WITH SECOND-SITUATION END-SITUATION-1))
**
(POSSESSION (WITH SELF POSSESSION-3)
            (WITH HAVER JOHN-1)
            (WITH OBJECT EXCHANGE-OBJECT-1)
            (WITH SITUATION END-SITUATION-1))
**
(POSSESSION (WITH SELF POSSESSION-4)
            (WITH HAVER MARY-1)
            (WITH OBJECT EXCHANGE-OBJECT-1)
            (WITH SITUATION BEGIN-SITUATION-1))
```

Note how only those frames in a hierarchy that are relevant to the subject-matter are instantiated. This is also an example where the same frame namely the one for possession-transfer gets instantiated several times.

2. 2. DOWNWARD MOVEMENT IN HIERARCHIES

We now go the other way, i.e. we specify information so that from a frame somewhere near the top of the hierarchy conditional expressions direct instantiations toward points lower in the hierarchy.

But conditionals are only part of the story. Suppose the reasoner is going down in the hierarchy and instantiates the frame for GIVE. When this frame becomes active it also wants to propagate its descriptions, so it causes the instantiation of the possession-transfer frame AGAIN ! If we are not careful here, the propagation would go in circles. The way to prevent this is to supply another piece of essential information to the reasoning system, namely the criteriality conditions. In the present example, if something is described as a possession transfer action, that same thing cannot again be described as a possession-transfer a moment later. Indeed if that is so we know we are talking about the same possession-transfer and the descriptions can be merged.

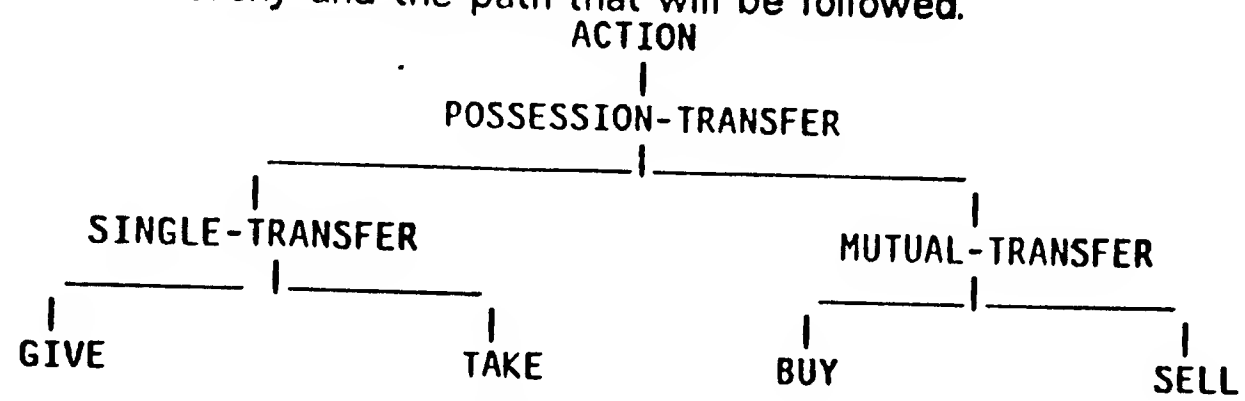
Another thing that will be demonstrated is the use of individual descriptions using the individuating specification, as in

```
(JOHN
  (WITH SELF)
  (ASPECT-SPECIFICATIONS:
   (INDIVIDUATING: (WITH-RESPECT-TO PEOPLE SELF))))
```

The reasoner keeps track of a list of the individuals that are accessible via individual descriptions. When the individual description occurs somewhere the instantiation process will try to find the corresponding object-expert instead of creating a new one. Note also that John is individuating with respect to people, i.e JOHN and (NOT MARY) would match if MARY was also declared to be individuating with respect to people.

The source description that will be given in the following example uses the possession-transfer frame and the hierarchy given earlier. The information that is necessary to move downward is that the old-owner has to be the same individual as the actor. We do this by specifying that the filler of both slots has the name JOHN. If the individuating-specifications are properly taken care of, the system should figure out that the fillers of these slots are the same.

Here is the hierarchy and the path that will be followed.



We start with the following transmission

```
>> (tell (an OBJECT)
         (you-are-described-as
          (A POSSESSION-TRANSFER
             (WITH ACTOR JOHN)
             (WITH OLD-OWNER JOHN)
             (WITH NEW-OWNER MARY)
             (WITH OBJECT (A BOOK)))))
```

Here are the resulting instantiations.

```
**
(POSSESSION-TRANSFER (WITH SELF POSSESSION-TRANSFER-1)
                     (WITH ACTOR JOHN-1)
                     (WITH OLD-OWNER JOHN-1)
                     (WITH NEW-OWNER MARY-1)
                     (WITH OBJECT OBJECT-1)
                     (WITH BEGIN-SITUATION BEGIN-SITUATION-1)
                     (WITH END-SITUATION END-SITUATION-1))
```

Observe that the old-owner has been identified as identical to the actor based on the individual-description attached to these slots.

```
**
(JOHN (WITH SELF JOHN-1))
**
(ACTION (WITH SELF POSSESSION-TRANSFER-1)
        (WITH ACTOR JOHN-1)
        (WITH BEGIN-SITUATION BEGIN-SITUATION-1)
        (WITH END-SITUATION END-SITUATION-1))
**
(MARY (WITH SELF MARY-1))
**
(BOOK (WITH SELF OBJECT-1))
**
(POSSESSION (WITH SELF POSSESSION-1)
            (WITH HAVER JOHN-1)
            (WITH OBJECT OBJECT-1)
            (WITH SITUATION BEGIN-SITUATION-1))
**
(TIME-SEQUENCE (WITH SELF TIME-SEQUENCE-1)
               (WITH FIRST-SITUATION BEGIN-SITUATION-1)
               (WITH SECOND-SITUATION END-SITUATION-1))
**
(POSSESSION (WITH SELF POSSESSION-2)
            (WITH HAVER MARY-1)
            (WITH OBJECT OBJECT-1)
            (WITH SITUATION END-SITUATION-1))
```

Downward movement now results in:

```
**
(GIVE (WITH SELF POSSESSION-TRANSFER-1)
      (WITH OBJECT OBJECT-1)
      (WITH OLD-OWNER JOHN-1)
      (WITH NEW-OWNER MARY-1)
      (WITH BEGIN-SITUATION BEGIN-SITUATION-1)
      (WITH END-SITUATION END-SITUATION-1))
```

Upward movement starts again from give. But no further action occurs because this instantiation merges with the first instantiation of possession-transfer.

```
**
(POSSESSION-TRANSFER (WITH SELF POSSESSION-TRANSFER-1)
                     (WITH ACTOR JOHN-1)
                     (WITH OLD-OWNER JOHN-1)
                     (WITH NEW-OWNER MARY-1)
                     (WITH OBJECT OBJECT-1)
                     (WITH BEGIN-SITUATION BEGIN-SITUATION-1)
                     (WITH END-SITUATION END-SITUATION-1))
```

This example shows clearly that we can start from anywhere in the hierarchy and move up and down - if sufficient information is there.

2. 3. MERGING OF OBJECTS

Now comes a more complicated example that uses the constraints on the frame to establish identity-links between object-experts.

First we introduce some new frames, in particular one for an individual with the name Ludwig, and an individual known as a blue-book that is possessed by Ludwig at a situation called yesterday

```
(BLUE-BOOK
   (WITH SELF
         (THE OBJECT OF A POSSESSION
                 (WITH HAVER LUDWIG)
                 (WITH SITUATION (A YESTERDAY)))))
```

Now we tell an object that it is the action of a possession-transfer where the actor is Ludwig, the object is the blue-book and the situation is the situation yesterday. We do not tell who the owner of the book is. Because the system knows that Ludwig owns the blue-book, it realizes that the old-owner and the actor of the possession-transfer are co-referential. It does this on the assumption that an object at a given situation has only one owner, i.e. the frame for POSSESSION has the following aspect-specifications

```
(POSSESSION
 (WITH SELF)
 (WITH HAVER)
 (WITH OBJECT)
 (WITH SITUATION)
 (ASPECT-SPECIFICATIONS:
  (CRITERIAL:
   (OBJECT SITUATION)))).
```

So, if you know the object and the situation then you have uniquely identified the instantiation.

Once it is realized that the old-owner and the actor of the possession-transfer are the same, identity-links have to be established between the two object-experts which were thought to be different at the time of instantiating the first description. Once this identity-link is established, the hierarchy can be expanded downward because now a clause in the conditional description attached to possession-transfer is known to be true. Here is a picture of what goes on:

```
>> (tell (an OBJECT)
         (you-are-described-as
          (A POSSESSION-TRANSFER
                  (WITH ACTOR LUDWIG)
                  (WITH NEW-OWNER MARY)
                  (WITH OBJECT (A BLUE-BOOK))
                  (WITH BEGIN-SITUATION (A YESTERDAY)))))
```

Note that we do not talk about the old-owner at all.

The following instantiations are created in response to this message.

```
**
(POSSESSION-TRANSFER (WITH SELF POSSESSION-TRANSFER-1)
                     (WITH ACTOR LUDWIG-1)
                     (WITH OLD-OWNER OLD-OWNER-1)
                     (WITH NEW-OWNER MARY-1)
                     (WITH OBJECT OBJECT-1)
                     (WITH BEGIN-SITUATION BEGIN-SITUATION-1)
                     (WITH END-SITUATION END-SITUATION-1))
```

An instantiation is made with the old-owner an anonymous object taken care of by OLD-OWNER-1. The actor of the transfer is called LUDWIG-1. So the old-owner and the actor are conceived as different objects at this time.

```
**
(LUDWIG (WITH SELF LUDWIG-1))
**
(YESTERDAY (WITH SELF BEGIN-SITUATION-1))
```

Upward movement !

```
**
(ACTION (WITH SELF POSSESSION-TRANSFER-1))
        (WITH ACTOR LUDWIG-1))
        (WITH BEGIN-SITUATION BEGIN-SITUATION-1)
        (WITH END-SITUATION END-SITUATION-1))
```

One constraint on POSSESSION-TRANSFER says that the old-owner must possess the object (here the blue-book, i.e. OBJECT-1) in the begin-situation (here BEGIN-SITUATION-1). The following instantiation is created in response to this constraint

```
**
(POSSESSION (WITH SELF POSSESSION-1)
            (WITH HAVER OLD-OWNER-1)
            (WITH OBJECT OBJECT-1)
            (WITH SITUATION BEGIN-SITUATION-1))
**
(MARY (WITH SELF MARY-1))
**
(BLUE-BOOK (WITH SELF OBJECT-1))
**
(TIME-SEQUENCE (WITH SELF TIME-SEQUENCE-1)
               (WITH FIRST-SITUATION BEGIN-SITUATION-1)
               (WITH SECOND-SITUATION END-SITUATION-1))
**
(POSSESSION (WITH SELF POSSESSION-2)
            (WITH HAVER MARY-1)
            (WITH OBJECT OBJECT-1)
            (WITH SITUATION END-SITUATION-1))
```

The instantiation of the blue-book frame yields the next constraint saying that Ludwig (here LUDWIG-1) is the owner of the blue-book (OBJECT-1) in the begin-situation (BEGIN-SITUATION-1).

```
**
(POSSESSION (WITH SELF POSSESSION-3)
            (WITH HAVER LUDWIG-1))
            (WITH OBJECT OBJECT-1)
            (WITH SITUATION BEGIN-SITUATION-1))
```

But we had already an instantiation of this frame and the haver in that instantiation was OLD-OWNER-1 ! Because of the criteriality of the object and situation aspect, i.e. if the

respective fillers of the object and situation aspects are identical the haver must also be identical, it becomes known that LUDWIG-1 and OLD-OWNER-1 are identical:

```
**
ESTABLISHED  IDENTITY OF LUDWIG-1 AND OLD-OWNER-1
```

Downward movement can now start:

```
**
(GIVE (WITH SELF POSSESSION-TRANSFER-1))
      (WITH OBJECT OBJECT-1)
      (WITH OLD-OWNER LUDWIG-1))
      (WITH NEW-OWNER MARY-1)
      (WITH BEGIN-SITUATION BEGIN-SITUATION-1)
      (WITH END-SITUATION END-SITUATION-1))
```

Upward from GIVE:

```
**
(POSSESSION-TRANSFER (WITH SELF POSSESSION-TRANSFER-1))
                     (WITH ACTOR LUDWIG-1))
                     (WITH OLD-OWNER LUDWIG-1))
                     (WITH NEW-OWNER MARY-1)
                     (WITH OBJECT OBJECT-1)
                     (WITH BEGIN-SITUATION BEGIN-SITUATION-1)
                     (WITH END-SITUATION END-SITUATION-1))
```

An attempt to match is now performed with the first instantiation of possession-transfer. Although the filler of the old-owner slot in this first instantiation is literally different from the old-owner in this instantiation (LUDWIG-1 vs. OLD-OWNER-1), a merge takes place because the matcher knows about identity of objects.

It is important to realize that insights such as identity of two objects are always obtained by a single expert and then distributed to the other experts which have to know about it. There is no active object except the expert !

2. 4. NEGATION

We now look an example that will illustrate how negation works. We will use the frames discussed earlier.

The example starts with telling an expert that it has a particular object at a certain time:

```
>> (tell (an OBJECT)
         (you-are-described-as
             (THE HAVER OF A POSSESSION)))
```

The possession frame is now instantiated and this will cause the creation of individuals for each of the roles in this frame, in particular the haver is treated by HAVER-1, the object by OBJECT-1 and the situation is SITUATION-1.

```
**
(POSSESSION (WITH SELF POSSESSION-1)
            (WITH HAVER HAVER-1)
            (WITH OBJECT OBJECT-1)
            (WITH SITUATION SITUATION-1))
```

Now we tell HAVER-1 that it is NOT the old-owner of a possession-transfer where the object is OBJECT-1 and the begin-situation is SITUATION-1. Our prediction of what will happen goes like this: A new anonymous individual is created and assigned to a particular expert. This new individual is specified as not being HAVER-1. The new expert then starts developing the consequences of being the old-owner in a possession-transfer situation. One of these consequences is that he is known to be the haver of the object in the begin-situation of the transfer. But we know that HAVER-1 is the owner. And because according to the definition an object can only have one owner at a given situation, HAVER-1 must be identical with this anonymous individual... So the system should arrive at a contradiction.

Here is what actually happens:

```
>> (tell HAVER-1
        (you-are-described-as
                (NOT (THE OLD-OWNER OF A POSSESSION-TRANSFER
                          (WITH OBJECT OBJECT-1)
                          (WITH BEGIN-SITUATION SITUATION-1)))))
**
(POSSESSION-TRANSFER (WITH SELF POSSESSION-TRANSFER-1)
                     (WITH ACTOR ACTOR-1)
                     (WITH OLD-OWNER OLD-OWNER-1)
                     (WITH NEW-OWNER NEW-OWNER-1)
                     (WITH OBJECT OBJECT-1)
                     (WITH BEGIN-SITUATION SITUATION-1)
                     (WITH END-SITUATION END-SITUATION-1))
```

Observe that the old-owner is a new expert called OLD-OWNER-1, instead of HAVER-1. Constraints are now propagated, in particular the constraint that there is a POSSESSION situation involving the old-owner, the begin-situation and the object. Based on this constraint the system figures out that OLD-OWNER-1 and HAVER-1 are identical (because criteriality of object and situation in POSSESSION-TRANSFER):

```
**
ESTABLISHED IDENTITY OF OLD-OWNER-1 AND HAVER-1
```

But now, just as we expected, there is a contradiction:

```
==>  CONTRADICTION IN HAVER-1 BETWEEN
(IS OLD-OWNER-1)
(IS (NOT OLD-OWNER-1))
```

3. DEVICES

The final example of this chapter deals with a completely different domain: algorithmic common sense reasoning. It illustrates more complicated applications of the mechanisms we have studied so far.

Algorithmic reasoning deals with mechanisms in the real world from the viewpoint of how they work. Reasoning about programs or electronic circuits are certain types of algorithmic reasoning. We look here at a more down-to-earth example: the notorious reverse-trap-flush-toilet. First we perform a conceptual analysis of the problem.

There is a number of different perspectives from which one can start looking at a mechanism like a pump or a flush-toilet.

ANATOMY: What are the components?
What is the physical arrangement of the components?
MECHANISM: How does it work?
What will happen to it if you do this or that?
USE: When to use it?
How to use it?

3. 1. ANATOMY

A flush-toilet has the following components: a tank, a pipe, a bowl-assembly and a release:

```
(REVERSE-TRAP-FLUSH-TOILET
   (WITH SELF)
   (WITH TANK)
   (WITH PIPE)
   (WITH BOWL-ASSEMBLY)
   (WITH RELEASE))
```

The object aspect allows us to talk about the object as a whole. There is a physical arrangement between these parts the specification of which is left as an exercise to the reader. What is more interesting is a description of the mechanics of the device.

3. 2. MECHANISM

It turns out that a toilet can be described as a decomposable system, so we will describe the mechanics of each of the parts and this will give us the operation of the whole.

The tank is a container filled with some liquid. But it is a tank with an interesting property: it fills itself when it is empty. How this filling happens will not concern us. (The fact that we can ignore details - or stated more precisely describe things at a certain level of detail without worrying about the rest is very important. This example is a good illustration how to go about doing this.)
The pipe is a device that connects two containers related by a release. If this release is released, the liquid of the first container flows into the second one.
The bowl-assembly is a container with a drain. When water is present in the container, it flows down the drain carrying whatever object that was located in the container to the drain.

So we introduce frames for each of these parts:

```
(SELF-FILLING-CONTAINER
  (WITH SELF)
  (WITH KIND-OF-CONTENT))

(CONTAINER-WITH-DRAIN
   (WITH SELF)
   (WITH CONTAINER-PART)
   (WITH DRAIN))

(CONNECTION-WITH-RELEASE
   (WITH SELF)
   (WITH RELEASE)
   (WITH SOURCE-CONTAINER)
   (WITH TARGET-CONTAINER))
```

and describe the whole in terms of the parts:

```
(REVERSE-TRAP-FLUSH-TOILET
    (WITH SELF)
    (WITH TANK
          (A SELF-FILLING-CONTAINER
             (WITH KIND-OF-CONTENT WATER)))
    (WITH BOWL-ASSEMBLY
          (THE CONTAINER-PART OF A CONTAINER-WITH-DRAIN))
    (WITH PIPE
          (THE CONNECTOR OF A CONNECTION-WITH-RELEASE
                    (WITH RELEASE (= THE-RELEASE))
                    (WITH SOURCE-CONTAINER (= THE-TANK))
                    (WITH TARGET-CONTAINER (= THE-BOWL-ASSEMBLY))))
    (WITH RELEASE)).
```

In other words, a flush-toilet is described as an object with four parts:

A tank which is described as a self-filling-container with kind-of-content the kind of water.

A bowl-assembly which is described as the container-part of a container-with-drain.

A pipe which is described as the connector of a connection-with-release with the release co-referential with the release of the toilet, the source-container co-referential with the tank and the target-container co-referential with the bowl-assembly.

A release which is the release of the pipe.

If we can specify the mechanics of the three devices used in the description then we have also specified the mechanics of the flush-toilet itself.

First we look at the self-filling-container, i.e. a kind of container that fills itself up when it is empty. To talk about this we need the notion of an empty-container:

```
(EMPTY-CONTAINER
  (WITH SELF)
  (WITH SITUATION)).
```

Also we need the notion of a liquid. Liquids are a very special class of objects and we will not discuss the complications for dealing with them here. For our present purposes we let a liquid be a concept with the following frame:

```
(LIQUID
  (WITH SELF)
  (WITH CONTAINER)
  (WITH KIND)
  (WITH SITUATION)).
```

So that we can talk about a certain amount of water as a certain object located in a certain container in a certain situation. Note that WATER can now be introduced as a kind of liquid:

```
(WATER
   (WITH SELF
         (A LIQUID))).
```

We will consider water to be an individual-concept.

Here is then the frame for the self-filling-container

```
(SELF-FILLING-CONTAINER
     (WITH SELF (= THE-CONTAINER)
           (WHEN THE-CONTAINER IS
                 ((AN EMPTY-CONTAINER
                      (WITH SITUATION (= A-SITUATION)))
                  (THE CONTAINER OF A LIQUID
                           (WITH SITUATION
                                 (THE SUCCESSOR OF A TIME-SEQUENCE
                                          (WITH PREDECESSOR
                                                (= A-SITUATION))))
                           (WITH KIND (= THE-KIND))))))
     (WITH KIND-OF-CONTENT (= THE-KIND))).
```

In other words, when the container is known to be an empty container in a certain situation, it can be deduced that the container will again contain a liquid of the kind it usually has at the situation immediately following the first situation

Note the frame for time-sequence which looks like this:

```
(TIME-SEQUENCE
  (WITH SELF)
  (WITH PREDECESSOR)
  (WITH SUCCESSOR)).
```

Time sequence is defined in such a way that situation which follow each other are identical, i.e. predecessor as well as successor are criterial.

Now the container with drain. This is somewhat more complicated. We know that as soon as some sort of liquid comes into the container part, it will flow down the drain. So we can describe the drain as the container of a liquid that was the moment before in the container-part.

```
(CONTAINER-WITH-DRAIN
     (WITH SELF)
     (WITH CONTAINER-PART)
     (WITH DRAIN
          (WHEN THE-CONTAINER-PART IS
               ((THE CONTAINER OF A LIQUID
                    (WHICH IS (= THE-LIQUID))
                    (WITH SITUATION (= THE-FIRST-SITUATION))
                    (WITH KIND (= THE-KIND)))
                (THE CONTAINER OF A LIQUID
                     (WHICH IS (= THE-LIQUID))
                     (WITH KIND (= THE-KIND))
                     (WITH SITUATION
                          (SUCCESSOR TIME-SEQUENCE
                               (WITH PREDECESSOR
                                    (= THE-FIRST-SITUATION)))))))))
```

But this is not enough. It is necessary to specify that when there is an object in the container-part, this object also ends up in the drain. To talk about this we need first a way to specify that an object is located at a place. We need two kinds of location specifiers: Being located on a certain surface:

```
(LOCATED-ON
  (WITH SELF)
  (WITH SUPPORT)
  (WITH OBJECT)
  (WITH SITUATION))
```

and being located in a certain container:

```
(LOCATED-IN
  (WITH SELF)
  (WITH CONTAINER)
  (WITH OBJECT)
  (WITH SITUATION)).
```

Now we add another conditional description, looking out for objects which might be located on the container-part of the bowl-assembly. When such an object is there it will also end up in the drain.

```
(CONTAINER-WITH-DRAIN
     (WITH CONTAINER-PART)
     (WITH DRAIN
          (WHEN THE-CONTAINER-PART IS
               ((THE CONTAINER OF A LIQUID
                    (WHICH IS (= THE-LIQUID))
                    (WITH SITUATION (= THE-FIRST-SITUATION))
                    (WITH KIND (= THE-KIND)))
                (THE CONTAINER OF A LIQUID
                    (WHICH IS (= THE-KIND))
                    (WITH OBJECT (= THE-LIQUID))
                    (WITH SITUATION
                         (AND
                          (THE SUCCESSOR OF A TIME-SEQUENCE
                              (WITH PREDECESSOR (= THE-FIRST-SITUATION)))
                          ;; looking out for the object
                          (WHEN THE-FIRST-SITUATION IS
```

```
                                ((THE SITUATION OF A LOCATED-ON
                                      (WITH OBJECT (= THE-OBJECT))
                                      (WITH SUPPORT (= THE-CONTAINER-PART)))
                                 (THE SITUATION OF A LOCATED-IN
                                      (WITH OBJECT (= THE-OBJECT))
                                      (WITH CONTAINER (= THE-DRAIN))))))))))))
```

In other words, the situation that is introduced as the situation where the drain contains the liquid is further constrained as a situation where the object is also in the drain when in the situation before that there was an object located in the container part. Note how complicated the co-referential links have become.

The final device is a connection-with-release. Such a device starts working when the release is put into action. Here is the RELEASE-ACTION frame structure:

```
(RELEASE-ACTION
  (WITH SELF)
  (WITH RELEASE)
  (WITH BEGIN-SITUATION
        (THE PREDECESSOR OF A TIME-SEQUENCE
                    (WITH SUCCESSOR (= THE-END-SITUATION))))
  (WITH END-SITUATION)).
```

What happens now is that when the release of a connection-with-release is the release of a release-action, the liquid that was in the source-container will flow to the target-container and the source-container will become empty:

```
(CONNECTION-WITH-RELEASE
   (WITH SELF)
   (WITH RELEASE)
   (WITH CONNECTOR)
   (WITH SOURCE-CONTAINER)
   (WITH TARGET-CONTAINER
         (WHEN THE-RELEASE IS
               ((THE RELEASE OF A RELEASE-ACTION
                     (WITH BEGIN-SITUATION (= THE-BEGIN-SITUATION))
                     (WITH END-SITUATION (= THE-END-SITUATION)))
                (WHEN THE-SOURCE-CONTAINER IS
                      ((THE CONTAINER OF A LIQUID
                            (WHICH IS (= THE-LIQUID))
                            (WITH SITUATION (= THE-BEGIN-SITUATION))
                            (WITH KIND (= THE-KIND)))
                       (THE-TARGET-CONTAINER IS
                            (THE CONTAINER OF A LIQUID
                                 (WHICH IS (= THE-LIQUID))
                                 (WITH KIND (= THE-KIND))
                                 (WITH SITUATION
                                       (THE SITUATION OF AN EMPTY-CONTAINER
                                            (WHICH IS
                                                  (= THE-SOURCE-CONTAINER))))))))))))
```

We are now ready to look at some examples. What we will do is cause the construction of a model and experiment with it.

The following message creates an object that is a reverse-trap-flush-toilet (note the use of a partial description).

```
>> (tell (an OBJECT)
         (you-are-described-as (A REVERSE-TRAP-FLUSH-TOILET)))
```

The first thing what happens is that object-experts are created for the device and each of the parts:

```
**
(REVERSE-TRAP-FLUSH-TOILET (WITH SELF REVERSE-TRAP-FLUSH-TOILET-1)
                           (WITH TANK TANK-1)
                           (WITH BOWL-ASSEMBLY BOWL-ASSEMBLY-1)
                           (WITH PIPE PIPE-1)
                           (WITH RELEASE RELEASE-1))
```

Also the parts are being instantiated:

```
**
(SELF-FILLING-CONTAINER (WITH SELF TANK-1)
                        (WITH KIND-OF-CONTENT WATER-1))
**
(CONTAINER-WITH-DRAIN (WITH SELF CONTAINER-WITH-DRAIN-1)
                      (WITH CONTAINER-PART BOWL-ASSEMBLY-1)
                      (WITH DRAIN DRAIN-1))
**
(CONNECTION-WITH-RELEASE (WITH RELEASE RELEASE-1)
                         (WITH CONNECTOR PIPE-1)
                         (WITH SOURCE-CONTAINER TANK-1)
                         (WITH TARGET-CONTAINER BOWL-ASSEMBLY-1))
**
(WATER (WITH SELF WATER-1))
```

The following message puts water in the tank:

```
>> (tell TANK-1
         (you-are-described-as
           (THE CONTAINER OF A LIQUID (WITH KIND WATER))))
```

which leads to the following instantiations:

```
**
(LIQUID (WITH SELF LIQUID-1)
        (WITH CONTAINER TANK-1)
        (WITH SITUATION SITUATION-1)
        (WITH KIND WATER-1)
```

The following message causes an anonymous object to be put in the bowl:

```
>> (TELL BOWL-ASSEMBLY-1 (you-are-described-as (THE SUPPORT OF A LOCATED-ON)))
```

which leads to

```
**
(LOCATED-ON (WITH SELF LOCATED-ON-1)
            (WITH SUPPORT BOWL-ASSEMBLY-1)
            (WITH OBJECT OBJECT-1)
            (WITH SITUATION SITUATION-2)).
```

So there is now an object in the container-part of the bowl-assembly at situation SITUATION-2, and there is water in the tank at the situation covered by SITUATION-1. The following message puts the device in motion by performing a release-action:

```
>> (tell RELEASE-1
        (you-are-described-as
           (THE RELEASE OF A RELEASE-ACTION
                        (WITH BEGIN-SITUATION SITUATION-1)
**                      (WITH END-SITUATION SITUATION-2))))
(RELEASE-ACTION (WITH SELF RELEASE-ACTION-1)
                (WITH RELEASE RELEASE-1)
                (WITH BEGIN-SITUATION SITUATION-1)
**              (WITH END-SITUATION SITUATION-2))
(TIME-SEQUENCE (WITH SELF TIME-SEQUENCE-1)
               (WITH SUCCESSOR SITUATION-2)
               (WITH PREDECESSOR SITUATION-1))
```

Liquid gets into the bowl-assembly in situation-2:

```
**
(LIQUID (WITH SELF LIQUID-1)
        (WITH CONTAINER BOWL-ASSEMBLY-1)
        (WITH SITUATION  SITUATION-2)
        (WITH KIND WATER-1))
```

The tank gets empty:

```
**
(EMPTY-CONTAINER (WITH SELF TANK-1)
                 (WITH SITUATION SITUATION-2))
```

Liquid gets into the drain in the next situation (SITUATION-3):

```
**
(LIQUID (WITH CONTAINER DRAIN-1)
        (WITH OBJECT LIQUID-1)
        (WITH SITUATION SITUATION-3)
        (WITH KIND WATER-1))
```

The tank is being filled up again in SITUATION-4:

```
**
(LIQUID (WITH SELF LIQUID-2)
        (WITH CONTAINER TANK-1)
        (WITH SITUATION SITUATION-4)
        (WITH KIND WATER-1))
**
(TIME-SEQUENCE (WITH SELF TIME-SEQUENCE-2)
               (WITH SUCCESSOR SITUATION-3)
               (WITH PREDECESSOR SITUATION-2))
```

Two successor situations exist for SITUATION-2 and they are merged:

```
**
ESTABLISHED  IDENTITY OF SITUATION-4 AND SITUATION-3
**
(TIME-SEQUENCE (WITH SUCCESSOR SITUATION-4)
               (WITH PREDECESSOR SITUATION-2))
```

Finally the object that was located in the bowl goes down the drain too:

```
**
(LOCATED-IN (WITH SELF LOCATED-IN-2)
            (WITH CONTAINER DRAIN-1)
            (WITH OBJECT OBJECT-1)
            (WITH SITUATION SITUATION-3))
```

Our toilet is now ready to be operated again. So let us cause a new release-action

without putting a new object into the bowl:

```
>> (tell RELEASE-1
         (you-are-described-as
          (THE RELEASE OF A  RELEASE-ACTION
               (WITH BEGIN-SITUATION SITUATION-4))))
```

And indeed, the device starts working again:

```
**
(RELEASE-ACTION (WITH SELF RELEASE-ACTION-2)
                (WITH RELEASE RELEASE-1)
                (WITH BEGIN-SITUATION SITUATION-4)
**              (WITH END-SITUATION SITUATION-5))
(TIME-SEQUENCE (WITH SELF TIME-SEQUENCE-3)
               (WITH SUCCESSOR SITUATION-5)
**             (WITH PREDECESSOR SITUATION-4))
(LIQUID (WITH SELF LIQUID-2)
        (WITH CONTAINER BOWL-ASSEMBLY-1)
        (WITH SITUATION SITUATION-5)
        (WITH KIND WATER-1)))
**
(EMPTY-CONTAINER (WITH SELF TANK-1)
**               (WITH SITUATION SITUATION-5))
(LIQUID (WITH SELF LIQUID-2)
        (WITH CONTAINER DRAIN-1)
        (WITH SITUATION SITUATION-6)
        (WITH KIND WATER-1))
**
(LIQUID (WITH SELF LIQUID-3)
        (WITH CONTAINER TANK-1)
        (WITH SITUATION SITUATION-7)
        (WITH KIND WATER-1))
**
(TIME-SEQUENCE (WITH SELF TIME-SEQUENCE-4)
               (WITH SUCCESSOR SITUATION-6)
**             (WITH PREDECESSOR SITUATION-5))
ESTABLISHED  IDENTITY OF SITUATION-6 AND SITUATION-7
**
(TIME-SEQUENCE (WITH SELF TIME-SEQUENCE-5)
               (WITH SUCCESSOR SITUATION-7)
               (WITH PREDECESSOR SITUATION-5))
```

3. 3. USE

The previous sections were devoted to a description of the toilet from the perspective of its components and from the perspective of the mechanics of these components. The final perspective we will look at is the perspective of how the device should be used. Here we envisage a hierarchy that would descend from ACTION and would contain several forms of WASTE-DISPOSAL. A frame for using a flush-toilet might look like this:

```
(PLAN-TO-USE-FLUSH-TOILET
  (WITH ACTOR)
  (WITH ACTION
        (A PLAN-TO-DISPOSE-WASTE
           (WITH ACTOR (= THE-ACTOR))
           (WITH WASTE (= THE-WASTE))))
  (WITH WASTE))
```

This gives us an upward pointer into the hierarchy of waste-disposal actions, and it is assumed that conditional descriptions attached to PLAN-TO-DISPOSE-WASTE will delimit other methods of waste-disposal to the one to be used with toilets.

Next we look into the problem of specifying how the device should be used. We all know how that goes: you put your waste into the bowl and then you start the release mechanism:

```
(PLAN-TO-USE-FLUSH-TOILET
  (WITH ACTOR)
  (WITH ACTION
        (A COMPOSITION-OF-TWO-ACTIONS
           (WITH FIRST-ACTION
                 (A PUT-IN
                    (WITH OBJECT (= THE-WASTE))
                    (WITH CONTAINER (= THE-BOWL-ASSEMBLY))
                    (WITH BEGIN-SITUATION (= THE-BEGIN-SITUATION))
                    (WITH END-SITUATION (= THE-SITUATION-IN-BETWEEN))))
           (WITH SITUATION-IN-BETWEEN
                 (= THE-SITUATION-IN-BETWEEN))
           (WITH SECOND-ACTION
                 (A RELEASE-ACTION
                    (WITH RELEASE (=THE-RELEASE))
                    (WITH BEGIN-SITUATION (= THE-SITUATION-IN-BETWEEN))
                    (WITH END-SITUATION (= THE-END-SITUATION))))))
  (WITH WASTE)
  (WITH BEGIN-SITUATION)
  (WITH END-SITUATION)
  (WITH TOILET
        (A REVERSE-TRAP-FLUSH-TOILET
           (WITH BOWL-ASSEMBLY (= THE-BOWL-ASSEMBLY))
           (WITH RELEASE (= THE-RELEASE))))
  (WITH BOWL-ASSEMBLY)
  (WITH RELEASE))
```

Here is the frame for composition of actions:

```
(COMPOSITION-OF-ACTIONS
   (WITH SELF)
   (WITH FIRST-ACTION)
   (WITH SITUATION-IN-BETWEEN)
   (WITH SECOND-ACTION))
```

and PUT-IN:

```
(PUT-IN
  (WITH SELF)
  (WITH ACTOR)
  (WITH OBJECT)
  (WITH CONTAINER
        (THE CONTAINER OF A LOCATED-IN
                    (WITH OBJECT (= THE-OBJECT))
                    (WITH SITUATION (= THE-END-SITUATION))))
  (WITH BEGIN-SITUATION)
  (WITH END-SITUATION)).
```

Suppose we now try to construct a model based on these descriptions. Then we would notice an interesting problem. First we put a certain object in the bowl using put-in. The object will now be in the bowl at a certain situation, say s-1 (in the frame s-1 is called the situation-in-between). Now the actor sets the toilet in motion by performing a release action. But there is a problem, how can the reasoner know that at the end of the release action (i.e. at the moment water starts getting into the bowl) the object is still in the bowl? The answer is because nothing happened that would change this state of affairs. But how does the reasoner know this?

In fact there is no simple answer to do this problem. We could try to attach some information causing the whole world to get updated each time something happened but that is clearly inadequate. This problem has a name. It is the frame-problem and we will see in a later chapter what we can do about it.

DISCUSSION

One principle of conceptual analysis that is used throughout is situational calculus developed by McCarthy (1959). The rain example comes from that paper.

The notion of a demon and its use in dealing with common sense tasks, such as understanding stories was first brought up in Charniak (1972). The example also illustrates that the conditional description is not only restricted to expressing hierarchies. Demons, but also generators (like in Conniver (cf. McDermott and Sussman,1973)) can be viewed as expressable with conditional descriptions.

Algorithmic common sense knowledge has been studied by Rieger (1975) and the reverse-trap-flush-toilet is his example. The example is also discussed in Smith (1978). Common sense knowledge in general is the main focus of work by Schank(1975), Wilks (1979), Hayes (1978), Martin (1979),a.o.

CONTROL

6. CONTROL

1. THE PRINCIPLES

1. 1. THE NECESSITY FOR CONTROL

The problem to be tackled in this chapter is the following: when a description is sent to a prototypical object it will be expanded until everything the reasoner can possibly know about this and related objects is part of the model.
Intuitively we feel that this is not right. The reasoner should develop only that portion of a model that is of current interest - in other words the *goals* should somehow determine how expansion proceeds. This intuitive judgment is justified by the following observations.

Recall that unbounded expansion reflected in the propagation of constraints principle, assumes that it will always be possible to modularize the domain such that a finite structure of questions, the constraint network, results. But now we observe that there are problem situations where this modularization is not sufficient, in other words, even though there is a finite network of questions it is not guaranteed that the processes will occur in finite time and space. These are situations where there is a series of alternative answers to a particular question but it is unknown in advance which alternative will work out.
In planning an action or proving a theorem in geometry for example, we may know several ways to get there without knowing which way will be successful. Or in language processing, a single word may have many different possible usages and to find out what usage applies in a particular situation can be a very complicated process.

Usually it is not possible to leave the alternatives alone until it can be decided which one is appropriate in the given situation because assumptions *have* to be made in order to derive any conclusions at all. This raises the question what laternative(s) should be investigated.

On the whole unbounded expansion of all possible alternatives is impossible because the accumulation of alternatives leads rapidly to so called *combinatorial explosions*. For example, if we estimate the number of usages of a word to be about 5 and if we have a sentence of 10 words, we would have to explore 5^{10} possibilities for that sentence. So how can we explain that a system can cope with such an enormous mass of accumulating alternatives?

A plausible answer to this puzzle is that such a system has the ability to represent and be guided by *strategies*. A typical example of such a strategy is 'depth-first search' : one alternative is developed and other alternatives are tried only if the first one failed. These strategies are often called heuristics. Heuristics allow us to control the combinatorial explosions caused by the accumulating effect of many alternatives because they contain advice on which alternative to explore and what to do if a certain alternative fails. Hence we postulate the following principle:

PRINCIPLE 14:
IT IS NECESSARY TO HAVE HEURISTIC KNOWLEDGE IN ORDER TO REASON ABOUT ALTERNATIVES.

1. 2. THE NEED FOR DOMAIN-SPECIFIC HEURISTICS

Let us develop this line of thought further and ask the question whether it is sufficient to have such a 'bag of methods' available that are put into effect when we deal with a problem - whatever its nature. Or whether it is necessary to have very specific domain-dependent knowledge about how to use knowledge within that domain. Under this second hypothesis there would be no set of golden rules that always work - instead each piece of knowledge is encapsulated in a network of advice specifying how to use this piece. Interestingly enough, this is again an instance of the general-special dilemma we encountered so often in previous chapters. If we can find some way of specifying heuristic knowledge that is domain-independent, we gain a higher degree of generality. If we need on the other hand domain-specific heuristic knowledge, the range of a particular advice is restricted.

It is this second line of thought which is now generally accepted. This is not for lack of trying to make the first hypothesis work. Numerous attempts have been made to construct systems based on a finite set of general purpose heuristics but results have been disappointing , in the sense that the control over the exploration of alternatives was insufficient. Intuitively it is clear that a strategy which works for one problem will not necessarily work for another problem, so that the use of a particular heuristic rule has to be tied up somehow with the nature of the problem situation and with the domain where this problem comes from. For example if we know that a certain alternative occurs in 99 percentage of the cases, then a reasonable strategy is to assume this alternative and only consider other alternatives when the first one fails. Which alternative is most likely to succeed can only be stated in the context of a particular subject-matter. That is the reason why we need domain-dependent heuristics, a principle expressed as follows:

PRINCIPLE 15:
IT IS NECESSARY TO HAVE DOMAIN-SPECIFIC HEURISTIC KNOWLEDGE ABOUT HOW TO ACTIVATE KNOWLEDGE ABOUT A DOMAIN

The next issue is how this heuristic knowledge should be represented.

1. 3. ON THE REPRESENTATION OF HEURISTIC KNOWLEDGE

Recall from earlier chapters that there are two modes of representation. Either we can represent knowledge declaratively or we can use procedures. Let us investigate what kind of a representation would be best for our present purposes.

When heuristic knowledge is represented procedurally, there would be a way to embed

these descriptions into procedures, for example by making a description the argument of a procedure call. These procedures would perform the process of keeping track how alternatives are developed.
An example of such a procedure might be ORDERED-ALTERNATIVES, as in,

```
(WORD-HAND
  (WITH SELF
        (ORDERED-ALTERNATIVES
                (A NOUN)
                (A VERB))))
```

which would mean that there are two alternatives for the category of the word hand. Either it could be a noun (as in 'A person has two hands') or a verb (as in 'he hands a letter to her'). Because the noun is more common, it would be explored first and if this usage fails, the reasoner has to back up and explore the next one in of a certain parent-child-relation. The first alternative (A NOUN) would be explored first and if that fails the reasoner has to back up and explore the next one in the series.
Clearly these procedures could be a lot more complicated, for example take arguments in the form of contextual information, etc. But the important idea is that actions 'behind the scene' would guide and control the exploration of alternatives.

On the other hand, when heuristic knowledge is represented declaratively we would first of all need concepts (or prototypes) dealing with issues of control. These concepts should allow us to express what possibilities have to be investigated, what should be done when a particular path fails, etc. As a result we would get two layers in the model: A layer that develops the details of the problem situation (that is what we had before) AND a layer that is a model of the problem solving behavior.
In contrast to the procedural solution, strategies and the way they are put to use would become explicitly available in the model. That is why this method is also called *explicit control of reasoning*.
For example, if we would deal with the problem of finding a path from one city to another one given a network of possible connections, the first model layer would explore knowledge of the cities, the connections, etc, whereas the second layer would keep track of what connections have been investigated already, what results have been obtained, what still needs to be done, and so on.

We will here argue for this second method of representing heuristic knowledge. The main argument is that the declarative solution is the most economical one in terms of additional mechanism needed to realize it and would therefore yield a stronger theory. In fact we do not have to expand the system at all because the descriptive apparatus used to represent knowledge about the domain would be the same as the one used to represent heuristic knowledge. As a consequence the mechanisms themselves would be the same too.

The main argument in favour of a procedural representation of heuristic knowledge is that this mode of representation leads to a stronger grip on the processes, i.e. that they lead to greater control. But if we can show that the declarative solution leads to sufficient control, this argument would become invalid.

Hence we adopt the following hypothesis on the way heuristic knowledge should be represented.

PRINCIPLE 16:

HEURISTIC KNOWLEDGE IS REPRESENTED DECLARATIVELY AND THE REASONER BUILDS UP A MODEL OF THE WAY IT SOLVES A PROBLEM IN ADDITION TO A MODEL OF THE PROBLEM SITUATION ITSELF.

Notice that this principle has interesting consequences for learning : One would become a better problem solver by acquiring control concepts, i.e. by finding better ways to keep track of his own problem solving behavior and by keeping a record why certain methods were successful in certain situations.

DISCUSSION

Problems with reasoning about alternatives dominated the first decade of research in AI. The first principle advanced here, namely the necessity of heuristic knowledge was recognized very soon and AI was therefore often called heuristic programming (cf. Minsky (1959), Feigenbaum and Feldman (1963) , a.o.). This principle is now generally recognized. On the other hand it took a long time before the second principle became accepted. Before that it was assumed that a certain set of heuristic tools could be found and that these tools could be applied to all cases. See e.g. McCarthy's proposal for the advice-taker (McCarthy,1959).

All this changed around 1970 when it was realized that heuristic knowledge needs to be domain specific. This viewpoint was first advanced within the procedural embedding of knowledge paradigm. Major proponents of this movement are Hewitt (1969,1975), Winograd (1972) ,a.o.

Soon after that the movement to make more and more knowledge explicit started to take shape. Consider e.g. the literature on planning. Early planning systems (like STRIPS, cf. Fikes and Nillson,1971), build a plan from the ground up each time it was needed. A further development consisted in introducing a library of methods or plans (cf. Fikes and Nilson (1972)). Planning now becomes a process of searching for a plan that more or less fits the goals of the problem situation and a candidate is then adjusted to the new situation. The next step consisted in introducing explicit representations of the reasoning process itself: both of heuristic rules to go about reasoning and of the intermediate steps and the goals. Early examples of this step are seen in Sacerdoti(1975), Hayes (1975), McDermott (1976) and Davis (1976). Sometimes this heuristic knowledge is phrased in terms of special purpose representations like ATN's (cf. Miller and Goldstein,1977). There is no need for this. The same knowledge structures used to represent and reason about the domain can as well be used to represent and reason over heuristics. (cf. DeKleer, et.al.,1977).

2. THE FRAMEWORK

An immediate consequence of the principle that heuristic knowledge will be represented declaratively and that reasoning will be controlled explicitly is that no real extension of the knowledge representation language or the reasoner is needed to obtain sufficient control. For example if we want to have a method that will show the equality between

two lists, then we can introduce a frame for show-list-equality and instantiate this frame whenever we want to see whether equality holds.

However the explicit invocation of method-frames becomes tedious in practical applications. We introduce therefore an implicit invocation mechanism. This mechanism is implemented by adopting the following strategy: whenever a match is attempted for a certain description and that description is not available, see whether there is a method-frame for the frame used in the description. Cause an instantiation of this frame. Because this frame will eventually turn up an answer, the matching will then proceed with whatever result that is desired.

For example, suppose we are working in the list-structures domain and we have a condition that checks whether two lists are equal. Then at this point a method to show-list-equality has to be invoked.

This algorithm requires that we are able to specify the relation between a given frame, e.g. the frame for equality, and its method-frame, e.g. the frame for show-list-equality. In order to do that we introduce a frame for METHOD:

```
(METHOD
  (WITH SELF)
  (WITH GOAL))
```

which will establish this relation for the reasoner. Thus we could have the following specification of the relation between equality and show-list-equality:

```
(LIST-EQUALITY
  (WITH SELF)
  (WITH SOURCE-LIST)
  (WITH TARGET-LIST))
```

and

```
(SHOW-LIST-EQUALITY
   (WITH SELF
         (A METHOD
            (WITH GOAL (A LIST-EQUALITY
                          (WITH SOURCE (= THE-SOURCE-LIST))
                          (WITH TARGET-LIST (= THE-TARGET-LIST))))))
   (WITH SOURCE-LIST)
   (WITH TARGET-LIST))
```

3. EXAMPLES

We will present two examples in this section to illustrate the problem solving paradigm defined by the principles in this chapter. These examples will illustrate

+ How knowledge can be encapsulated in control descriptions so that it is only expanded when needed.
+ How one can represent the state of the problem solver and let these descriptions influence the reasoning process.
+ How alternatives can be represented in a way that allows the expression of heuristic knowledge
+ How results can be expressed in a form suitable for the user

+ How frames for methods can be constructed.

The first example is relatively straightforward. It comes from the domain of musical reasoning. The example does illustrate the indirect invocation of method-frames. The second example is well-known in the literature as a typical search problem: finding the connection between two cities given a network of connections. This example concentrates on explicit control of reasoning.

3. 1. THE PASSING-CHORD PROBLEM

Let us look at the following musical problem which comes from tonal harmony. It consists in finding a chord (the so called passing-chord) between two other chords given a set of constraints for going from one to the other. In the present example, the constraint will be as follows:

(i) The interval between the root of the first chord and the root of the passing-chord is either a fourth, a fifth, a halfstep up or a halfstep down.
(ii) The interval between the passing-chord and the second chord is also either a fourth, a fifth, a halfstep up or a halfstep down.

For example, suppose the root of the first chord is C and the root of the second chord is D, then C-sharp is a root for a possible passing chord because there is a half-step interval between C and C-sharp and another half-step between C-sharp and D. But G is also a possible solution because there is a fifth-interval between C and G: A fifth means (at least in the context of this example) three whole-steps and a half-step which is the case because we have

C -whole-step-> D -whole-step-> E -whole-step-> F-sharp -half-step-> G.
The second constraint holds for G as well because there is another fifth interval between G and D:
G -whole-step-> A -whole-step-> B -whole-step-> C-sharp -half-step-> D.

Let us first introduce frames for the concepts fourth, fifth, halfstep and whole-step. These concepts each have slots for the root (the start-tone of the interval) and the end-tone. A certain root as only half-step, whole-step, fifth or fourth and a given end-tone is the end-tone is only one half-step, whole-step, fourth or fifth. So root and end-tone are both criterial in each frame.

```
(HALF-STEP
  (WITH SELF)
  (WITH ROOT)
  (WITH END-TONE))
(WHOLE-STEP
  (WITH SELF)
  (WITH ROOT)
  (WITH END-TONE))
```

```
(FOURTH
  (WITH SELF)
  (WITH ROOT)
  (WITH END-TONE))
(FIFTH
  (WITH SELF)
  (WITH ROOT)
  (WITH END-TONE))
```

Next we need frames for each of the twelve tones in the tone-structure: C, C-sharp, D, D-sharp or E-flat, etc. This tone-structure can be viewed as a ring based on half-step relations. The tones are the primitive objects of the domain and the halfstep-relation is the most primitive relation between these objects:

```
(C
   (WITH SELF
         (ROOT HALF-STEP
               (WITH END-TONE C-SHARP))))

(C-SHARP
   (WITH SELF
         (ROOT HALF-STEP
               (WITH END-TONE D))))

 ...

(B
   (WITH SELF
         (ROOT HALF-STEP
               (WITH END-TONE C))))
```

Each tone is an individuating-concept with respect to other tones. In other words each tone has the following aspect-specifications:

```
(ASPECT-SPECIFICATIONS:
  (INDIVDUATING: (WITH-RESPECT-TO TONES SELF)))
```

Observe that when we would create a frame for any tone in the ring, which will happen as soon as a tone is mentioned in a description, frames for all tones will be created. This is justified because tones will be relevant to almost all the music problems one might be interested in. Because the tone-aspect of each tone is an individuating aspect, these tones will be created only once for every possible application.

Because whole-step relations are equally important, we incorporate the whole-step relations also in each model. These whole-step relations can be deduced from half-step relations as follows:

```
(HALF-STEP
     (WITH SELF)
     (WITH END-TONE)
     (WITH ROOT
          (IF THE-END-TONE IS
               ((THE ROOT OF A HALF-STEP
                    (WITH END-TONE (= ANOTHER-END-TONE)))
                ;; then the root is
                (THE ROOT OF A WHOLE-STEP
                     (WITH END-TONE (= ANOTHER-END-TONE))))))
```

Note that the whole-step relations will also form a ring structure because the half-step relations do.

So far there is not much difference with the way we would have done things in earlier chapters. But now we start to get more careful. Instead of generating all fourths and fifths all the time, we will only construct them when needed. In order to accomplish this we introduce method frames which try to establish a certain relation in the model. These frames are then instantiated, by performing a predication which makes reference to the frame. Note that by making the aspects of the method frame criterial, it will only be instantiated once, i.e. we have a sort of 'memo-ization' process going on.

Let us start with a frame to find the fourth. A fourth requires 2 whole-steps and a half-step:

```
(FIND-FOURTH
   (WITH SELF)
   (WITH ROOT (= THE-ROOT)
        (IF THE-ROOT IS
             ((THE ROOT OF A WHOLE-STEP
                  (WITH END-TONE (= THE-FIRST-END-TONE)))
              (IF THE-FIRST-END-TONE IS
                   ((THE ROOT OF A WHOLE-STEP
                        (WITH END-TONE (= THE-SECOND-END-TONE)))
                    (IF THE-SECOND-END-TONE IS
                         ((THE ROOT OF A HALF-STEP
                              (WITH END-TONE
                                   (= THE-FOURTH)))
                          (THE ROOT OF A FOURTH
                              (WITH END-TONE (= THE-FOURTH))))))))))))
```

So the reasoner will first look out whether the root of the tone from which we try to find a fourth interval, is the root of a whole-step. If that is so the reasoner will try to find another whole-step starting from the end-tone of this first whole-step. Once this is found it tries to find a half-step from the end-tone of the second whole-step. The end-tone of this half-step is the end-tone of the fourth from the initial root.

There are two ways in which this frame could be instantiated. Either directly, for example by describing a particular tone as the root of a find-fourth frame, or indirectly. In order to get the second type of invocation, we have to describe the find-fourth frame as a method to find the fourth of a given root:

```
(FIND-FOURTH
   (WITH SELF
         (A METHOD
            (WITH GOAL (A FOURTH
                           (WITH ROOT (= THE-ROOT))))))
   (WITH ROOT
         (IF THE-ROOT IS
               ((THE ROOT OF A WHOLE-STEP
                     (WITH END-TONE (= THE-FIRST-END-TONE)))
                (IF THE-FIRST-END-TONE IS
                      ((THE ROOT OF A WHOLE-STEP
                            (WITH END-TONE (= THE-SECOND-END-TONE)))
                       (IF THE-SECOND-END-TONE IS
                            ((THE ROOT OF A HALF-STEP
                                  (WITH END-TONE
                                        (= THE-FOURTH)))
                             (THE ROOT OF A FOURTH
                                  (WITH END-TONE (= THE-FOURTH)))))))))))
```

A fifth requires 3 whole steps and one 1/2 step. We will use the frame to compute a fourth first as 'sub-routine'

```
(FIND-FIFTH
   (WITH SELF
         (A METHOD
            (WITH GOAL (A FIFTH
                           (WITH ROOT (= THE-ROOT))))))
   (WITH ROOT
         (IF THE-ROOT IS
             ((THE ROOT OF A FOURTH
                  (WITH END-TONE (= THE-FOURTH)))
              ;; now one whole step further
              (IF THE-FOURTH IS
                  ((THE ROOT OF A WHOLE-STEP
                        (WITH END-TONE (= THE-FIFTH)))
                   (THE ROOT OF A FIFTH
                        (WITH END-TONE (= THE-FIFTH)))))))))
```

Observe how we described the root for which we want to find the fifth-interval as the root of a find-fourth frame. This will cause an instantiation of the find-fourth frame and will eventually result in the predication of a particular fourth relation between this root and a certain end-tone. The frame for find-fifth then picks up this fourth relationship and tries to find a further whole-step.

Now we start with frames for the passing-chord problem itself. The result should be a description referring to the following frame:

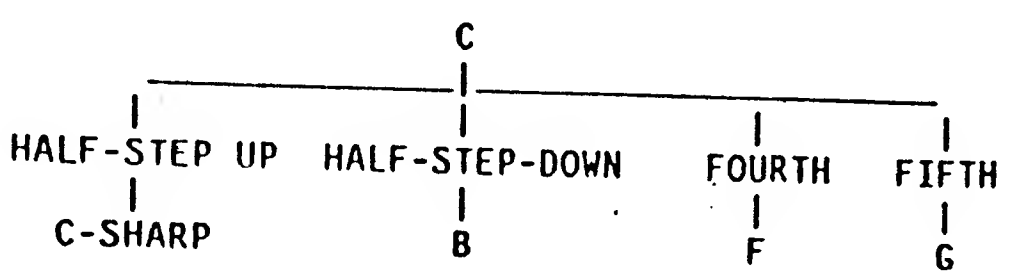
```
(POSSIBLE-PASSING-CHORD
        (WITH SELF)
        (WITH START-TONE)
        (WITH PASSING-CHORD)
        (WITH END-TONE))
```

To find such a description we use two frames: one that finds a possible passing chord and one that finds the first step for a passing chord, a concept with the following frame

```
(POSSIBLE-STEP
  (WITH SELF)
  (WITH START)
  (WITH END))
```

The method for finding a passing chord goes as follows: Try to find a possible step. If you have found one perform the second test. If this second test succeeds you have a valid result.

In other words we perform basically a breadth-first parallel search. All steps are generated for going from the start-tone to a possible passing chord by the find-step frame and the find-passing-chord frame filters the solutions that come out in order to find the ones that satisfy the second constraint. outcome of the find-step frame can be visualized as a tree, as the following one



The leaves of this tree are then investigated by the FIND-PASSING-CHORD frame.

Here is the method frame to find the first step. Then there is a conditional that will trigger on any interval that satisfies one of the following constraints: a fourth, a fifth, a half-step up or half-step down. If any of these intervals is found, the start-tone is described as the start of a possible step with as end the other element in the interval relation:

```
(FIND-STEP
    (WITH SELF
          (A METHOD
             (WITH GOAL (A POSSIBLE-STEP (WITH START (= THE-START-TONE))))))
    (WITH START-TONE
          (IF THE-START-TONE IS
              ((OR (THE ROOT OF A FOURTH
                        (WITH END-TONE (= THE-TONE)))
                   (THE ROOT OF A FIFTH
                        (WITH END-TONE (= THE-TONE)))
                   (THE ROOT OF A HALF-STEP
                        (WITH END-TONE (= THE-TONE)))
                   (THE END-TONE OF A  HALF-STEP
                        (WITH ROOT (= THE-TONE))))
               (THE START OF A POSSIBLE-STEP
                    (WITH END (= THE-TONE)))))))
```

Note how the same concept is used for half-step up and half-step down.

Finally here is the method frame for the passing-chord itself:

```
(FIND-PASSING-CHORD
    (WITH SELF
          (A METHOD
             (WITH GOAL
```

```
                    (A PASSING-CHORD
                       (WITH START-TONE (= THE-START-TONE))
                       (WITH END-TONE (= THE-END-TONE))))))
(WITH START-TONE)
(WITH END-TONE
      (IF THE-START-TONE IS
          ((THE START OF A POSSIBLE-STEP
                (WITH END (= THE-TONE-IN-BETWEEN)))
           (IF THE-TONE-IN-BETWEEN IS
               ((OR (THE ROOT OF A FOURTH
                         (WITH END-TONE (= THE-END-TONE)))
                    (THE ROOT OF A FIFTH
                         (WITH END-TONE (= THE-END-TONE)))
                    (THE ROOT OF A HALF-STEP
                         (WITH END-TONE (= THE-END-TONE)))
                    (THE END-TONE OF A HALF-STEP
                         (WITH ROOT (= THE-END-TONE))))
                (THE END-TONE OF A POSSIBLE-PASSING-CHORD
                     (WITH PASSING-CHORD (= THE-TONE-IN-BETWEEN))
                     (WITH START-TONE (= THE-START-TONE))))))))
```

These frames should demonstrate clearly that it is possible to introduce tight control over what exactly will be expanded in a given model. Rather than generate all possible fourths or all possible fifths, only those will be generated that are relevant to the problem we are dealing with.

In the next chapter we will have a conversation with the reasoner where these farmes are being used.

Readers familiar with programming must have noticed that method frames do not really differ from procedural specifications of tasks. Indeed a LISP program could be written that behaves in exactly the same way as the search example discussed here. What we seem to have here is a sort of synthesis of declarative and procedural ways of specifying processes. But whereas another synthesis (in particular the ACTOR-theory) was made by viewing everything as a procedure, we view everything as a description. The examples discussed here should make it clear that this does not go at the expense of control. Based on this new synthesis we suddenly see how there are declarative analogues for well known procedural notions, compare

PROCEDURE - FRAME
ARGUMENTS - ASPECTS
EVALUATION - INSTANTIATION
PROCEDURE CALL - ATTACHMENT
VARIABLE - OBJECT
VALUE - DESCRIPTION
ASSIGNMENT - PREDICATION
BINDING - ESTABLISHING CO-REFERENTIAL LINKS

etc.

4. EXPLICIT CONTROL OF REASONING

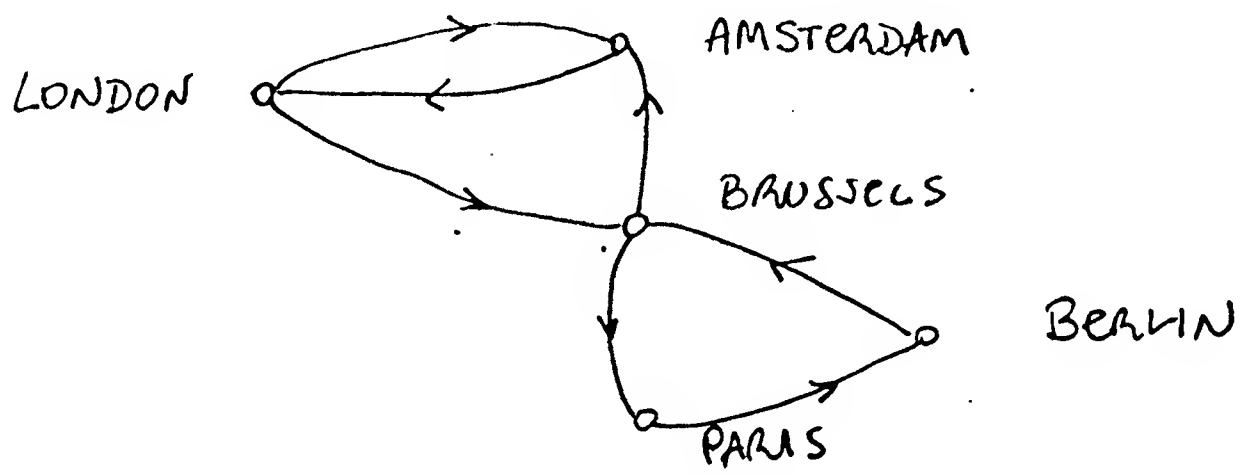
Now comes a second example, concentrating on the principle that control concepts should be introduced and that deduction should be controlled by constructing a model of the problem solving process at the same time as a model of the problem situation. We will do this in the context of an example which has been used in the literature to illustrate the use of heuristic knowledge: Find a path through a network of points given a certain initial point and a final destination. This problem will be phrased in terms of air-line connections.

Warning! This example is quite extensive. An impatient reader may skip it because no new material is introduced that would be necessary for further chapters.

4. 1. THE PROBLEM

Assume there are 6 cities: London, Amsterdam, Brussels, Paris and Berlin which are connected by certain air-line connections in the following way:

You can go from London to Amsterdam or Brussels, from Amsterdam only to London, from Brussels to Amsterdam or Paris, from Paris to Berlin and from Berlin to Brussels.



Although this network looks simple, there is plenty of opportunity to get in trouble with an unbounded expansion. For example one can go in circles such as between London and Amsterdam and back. It is of course possible to construct a fairly straightforward solution for this problem, that would however refute the purpose of this example. Instead we will introduce as much control concepts as possible.

4. 2. ENCAPSULATING KNOWLEDGE

The first measure of control we will introduce is this: a certain piece of knowledge will be encapsulated in a broader description in such a way that the item in question only becomes active when it is needed.

This can be realized by introducing a conditional description each time a certain fact is potentially useful (but not always). The conditions of this conditional description are

control descriptions. When one of them holds, the pending resulting-description will be released. One can think about this in the following way: Rather than always be available, each fact needs to be asked for before it becomes consultable.

Here is an example illustrating this method. It is not necessary to bring all possible connections from one city to another in the model as soon as we create an object for a city. In fact this would cause the creation of the whole network as soon as we formulate even a simple request. Instead connections should be encapsulated in a control conditional and only be released when needed.

First we introduce some frames. A frame for a city, like LONDON, will look like this:

```
(LONDON
  (WITH SELF)
  (ASPECT-SPECIFICATIONS:
   (INDIVIDUATING: SELF)))
```

so that we can talk about the object which is the city of london as

```
  LONDON
```

and that we can attach descriptions (e.g. the possible connections) to the SELF-slot in the LONDON frame. Note that LONDON is an individual concept, so that there will be only one expert in the model which can be described as the city of London.

Also we need some frames for representing connections. Let us therefore introduce a frame for the notion of POSSIBLE-CONNECTION:

```
(POSSIBLE-CONNECTION
  (WITH SELF)
  (WITH POINT-OF-DEPARTURE)
  (WITH DESTINATION)),
```

so that we can describe the city of london as follows:

```
(LONDON
  (WITH SELF
        (THE POINT-OF-DEPARTURE OF A POSSIBLE-CONNECTION
                  (WITH DESTINATION AMSTERDAM))))
```

In other words, London is described as the point-of-departure of a possible-connection with Amsterdam the destination.

In order to obtain greater control, we can make the release of this piece of information subject to a particular state of the problem solver, i.e. the problem solver must be interested in the connections before London will be described in terms of this description. This can be done by embedding the description attached to the self-slot of LONDON into a structure containing advice on when it should be used. In order to do this we first introduce the concept of 'ATTENTION'. When an object is under attention by the problem solver we will describe it as the focus of attention

```
(ATTENTION
  (WITH SELF)
  (WITH FOCUS))
```

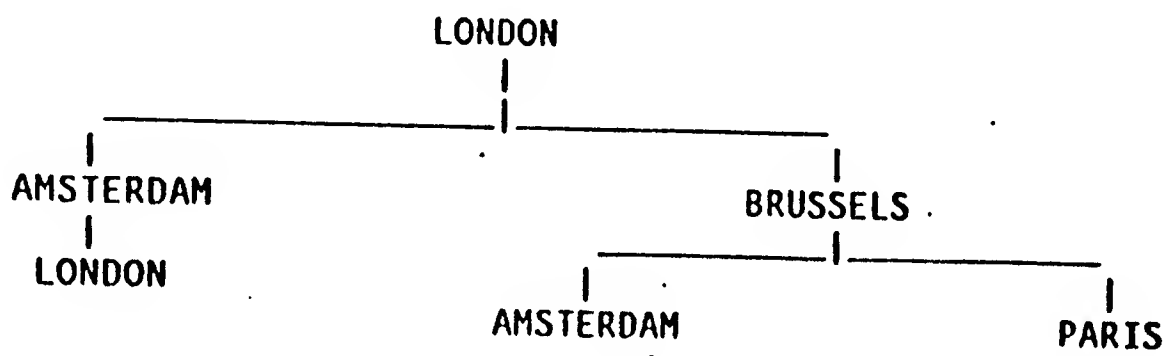
Now we can say

```
(LONDON
  (WITH SELF (=  THE-CITY)
        (IF THE-CITY IS
            ((THE FOCUS OF AN ATTENTION)
               (THE POINT-OF-DEPARTURE OF A POSSIBLE-CONNECTION
                    (WITH DESTINATION AMSTERDAM))))))
```

When a certain expert now wants to know what the possible-connections are going from the city of London, it will have to describe LONDON as the focus of attention. As soon as that happens the attached descriptions will be released.

4. 3. CONCEPTS DESCRIBING THE STATE OF THE PROBLEM SOLVER

The second tool for controlling the exploration of alternatives consists in the introduction of concepts describing the state of the problem solver. Typically for a search through a network such a state can be viewed as a tree structure. The following structure gives us a possible search path for going from London to Paris:



From London one can go to Amsterdam or Brussels. Suppose we take a flight to Amsterdam. The only possible connection from there is back to London. This brings us to the point of departure and we conclude that this is therefore a bad route. The other possibility is to go to Brussels. There are two possible connections from that city. Let us explore the first one. This brings us to Amsterdam. But we have been there already and we know that this did not lead to our final destination, hence we back up and take the other possibility which brings us to Paris which is where we have to be.

This method of search is known as depth-first search: the first alternative in the list is explored and if that fails the next alternative in the list is tried. There are many other methods possible and they can all be represented. But let us pursue this particular method for the sake of the example.

In order to do this we need two things: frames expanding the tree if needed and frames recording what route is taken and what cities have been visited so that we know what to do if a back-up occurs and can prevent that a particular city is visited twice. Here are some frames that will enable us to do so.

The following frame introduces the ability to describe a city as having been visited on a previous route:

```
(PREVIOUS-VISIT
 (WITH SELF)
 (WITH CITY)
 (WITH ROUTE))
```

A route corresponds to a particular portion of the investigation. Routes are related in the sense that a certain route can be the sub-route of another one.

```
(ROUTE
  (WITH SELF)
  (WITH SUB-ROUTE)
  (WITH SUPER-ROUTE))
```

For example the route for going from London to Brussels will have two sub-routes: one for going from Brussels to Amsterdam and one for going from Brussels to Paris.

We now observe the following relationships: when a certain route is the sub-route of another one, then any sub-routes of this route are also sub-routes of the other one. In other words relationships between routes are transitive:

```
(ROUTE
  (WITH SELF)
  (WITH SUB-ROUTE
        (IF THE-SUPER-ROUTE IS
            ((THE SUB-ROUTE OF A ROUTE
                  (WITH SUPER-ROUTE (= ANOTHER-ROUTE)))
             (THE SUB-ROUTE OF A ROUTE
                  (WITH SUPER-ROUTE (=.ANOTHER-ROUTE))))))
  (WITH SUPER-ROUTE))
```

Also being a city on a previous visit carries through via the sub-route relationships, in the sense that when a city was visited on a certain route and this route is the super-route of another one, then the city was also visited on this other route:

```
(PREVIOUS-VISIT
  (WITH SELF)
  (WITH CITY
        (IF THE-ROUTE IS
            ((THE SUPER-ROUTE OF A ROUTE
                  (WITH SUB-ROUTE (= THE-SUB-ROUTE)))
             (THE CITY OF A PREVIOUS-VISIT
                  (WITH ROUTE (= THE-SUB-ROUTE))))))
  (WITH ROUTE))
```

Descriptions formed on the basis of these frames will allow us to express information such as when the city has been visited already on this particular route, continue with the next possibility rather than pursueing this route further.

4. 4. REPRESENTING ALTERNATIVES

Alternatives are expressed explicitly, for example as a list of possibilities, and control concepts are introduced protecting a certain alternative from becoming active when it is not necessary.

Here is an example of this method. Let us represent a list of possible-connections by

way of a frame like the following one:

```
(POSSIBLE-CONNECTIONS
  (WITH SELF)
  (WITH POINT-OF-DEPARTURE)
  (WITH FIRST-POSSIBILITY)
  (WITH OTHER-POSSIBILITIES))
```

The list of 'other-possibilities' could be empty, in which case we make reference to a frame for the EMPTY-LIST:

```
(EMPTY-LIST
   (WITH SELF))
```

Now we can describe the information that one can go from London to Amsterdam *or* Brussels as follows:

```
(LONDON
  (WITH SELF (= THE-CITY)
    (IF THE-CITY IS
     ((THE FOCUS OF AN ATTENTION)
      (THE POINT-OF-DEPARTURE OF A POSSIBLE-CONNECTIONS
           (WITH FIRST-POSSIBILITY
                 (A POSSIBLE-CONNECTION
                    (WITH DESTINATION AMSTERDAM)
                    (WITH POINT-OF-DEPARTURE (= THE-CITY))))
           (WITH OTHER-POSSIBILITIES (= THE-OTHER-POSSIBILITIES)
                 (IF THE-OTHER-POSSIBILITIES IS
                     ((THE FOCUS OF AN ATTENTION)
                      (THE-CITY IS
                          (THE POINT-OF-DEPARTURE OF A POSSIBLE-CONNECTIONS
                               (WITH FIRST-POSSIBILITY
                                    (A POSSIBLE-CONNECTION
                                       (WITH DESTINATION PARIS)
                                       (WITH POINT-OF-DEPARTURE (= THE-CITY))))
                               (WITH OTHER-POSSIBILITIES (AN EMPTY-LIST)))))))))))
```

The reader might want to construct at this point frames for the other cities in the network.

4. 5. COLLECTING THE RESULT

When the development of a model is not goal-directed, it is unclear what the result should be. However when we have a particular goal in mind one can introduce frames that will look out for this goal and express the information in a way one wants it to have. Let us give some examples here.

Suppose we want the final result to be expressed by frames specifying whether there is a connection between two cities, and if so what exactly this connection is. Let us say that a specific connection is expressed as follows

```
(RECOMMENDED-CONNECTION
  (WITH SELF)
  (WITH POINT-OF-DEPARTURE)
  (WITH DESTINATION))
```

whereas a possible-route between two points is expressed like this

```
(POSSIBLE-ROUTE
  (WITH SELF)
  (WITH CONNECTION)
  (WITH POINT-OF-DEPARTURE)
  (WITH DESTINATION))
```

If we now know that there is a possible path from a certain point-of-departure to a certain final-destination for a given possible-connection, we can express the information in the way we want it as follows:

```
(POSSIBLE-PATH
    (WITH SELF)
    (WITH POSSIBLE-CONNECTION
          (AND
           (THE CONNECTION OF A POSSIBLE-ROUTE
                (WITH POINT-OF-DEPARTURE (= THE-POINT-OF-DEPARTURE))
                (WITH FINAL-DESTINATION (= THE-FINAL-DESTINATION)))
           (A RECOMMENDED-CONNECTION
              (WITH POINT-OF-DEPARTURE (= THE-POINT-OF-DEPARTURE))
              (WITH DESTINATION (= THE-FINAL-DESTINATION)))))
    (WITH POINT-OF-DEPARTURE)
    (WITH FINAL-DESTINATION))
```

In other words the possible connection is described as the connection of a possible route and a recommended connection.

Here is a more complicated 'wrap up' of results which would be useful if we have already developed a partial path and we want to *cons* a new connection in front of the path:

```
(RECOMMENDED-PATH
    (WITH SELF)
    (WITH POSSIBLE-CONNECTION
          (AND
           (THE FIRST-CONNECTION OF A COMBINATION-OF-CONNECTIONS
                 (WITH SECOND-CONNECTION
                       (= THE-OTHER-CONNECTIONS)))
           (THE CONNECTION OF A RECOMMENDED-CONNECTION
                (WITH POINT-OF-DEPARTURE (= THE-POINT-OF-DEPARTURE))
                (WITH DESTINATION (= THE-DESTINATION)))
           (THE CONNECTION OF A POSSIBLE-ROUTE
                (WITH POINT-OF-DEPARTURE (= THE-POINT-OF-DEPARTURE))
                (WITH FINAL-DESTINATION (= THE-FINAL-DESTINATION)))))
    (WITH POINT-OF-DEPARTURE)
    (WITH FINAL-DESTINATION)
    (WITH DESTINATION)
    (WITH OTHER-CONNECTIONS))
```

What this frame does is add a connection (the-possible-connection) to an existing list of connections (called the-rest-of-the-connections).

4. 6. METHOD FRAMES

The next tool realizing the explicit control of reasoning paradigm consists in introducing method frames, i.e. frames which direct the problem solving process in certain ways depending on descriptions of what has been achieved, what still needs to be done, etc. Here are some examples of such frames.

First we introduce a frame for the investigation of the next possibility of a list of possibilities with aspects for the possibilities, the final-destination and the point-of-departure. This frame contains the following method: when there are no possibilities left, describe the point-of-departure as the point-of-departure of a bad-connection for that particular final-destination. Otherwise describe the other possibilities as being under consideration:

```
(INVESTIGATION-OF-NEXT-POSSIBILITY
    (WITH SELF)
    (WITH FINAL-DESTINATION)
    (WITH POINT-OF-DEPARTURE)
    (WITH OTHER-POSSIBILITIES
          (IF-NOW THE-OTHER-POSSIBILITIES IS
             ((AN EMPTY-LIST)
              (THE-POINT-OF-DEPARTURE IS
                     (THE POINT-OF-DEPARTURE OF A BAD-CONNECTION
                          (WITH FINAL-DESTINATION (= THE-FINAL-DESTINATION)))))
             (ELSE
              (THE-OTHER-POSSIBILITIES IS
                        (THE FOCUS OF AN ATTENTION))))))
```

where the frame for a 'bad connection' has the following skeleton:

```
(BAD-CONNECTION
(WITH SELF)
(WITH POINT-OF-DEPARTURE)
(WITH FINAL-DESTINATION))
```

Here is another example. Suppose we have arrived at a certain city which is not equal to the final destination, nor a city previously visited. Let us call such a city a possible-destination. Now we would like to investigate whether you can go from this city to the final-destination. A method for doing that might go as follows: unless it is known that this possible-destination does not lead to the final-destination, i.e. that it is the point-of-departure of a bad-connection, describe the path that was being followed as the path of a journey where this possible-destination is the point-of-departure, the route is a sub-route of the route followed until now and the final destination is the original final-destination :

```
(FURTHER-INVESTIGATION
    (WITH SELF
    (WITH POINT-OF-DEPARTURE)
    (WITH FINAL-DESTINATION)
    (WITH ROUTE)
    (WITH PATH)
    (WITH POSSIBLE-DESTINATION
          (IF-NOW THE-POSSIBLE-DESTINATION IS
              ((THE POINT-OF-DEPARTURE OF A BAD-CONNECTION
                    (WITH FINAL-DESTINATION (= THE-FINAL-DESTINATION)))
               ;; just a dummy description
               (= THE-POSSIBLE-DESTINATION))
              (ELSE
               (THE-PATH IS
                  (THE PATH OF A JOURNEY
                       (WITH POINT-OF-DEPARTURE (= THE-POSSIBLE-DESTINATION)
                             (THE CITY OF A PREVIOUS-VISIT
                                  (WITH ROUTE (= THE-SUB-ROUTE))))
                       (WITH FINAL-DESTINATION (= THE-FINAL-DESTINATION))
                       (WITH ROUTE (= THE-SUB-ROUTE)
                             (THE SUB-ROUTE OF A ROUTE
                                  (WITH SUPER-ROUTE (= THE-ROUTE)))))))))))
```

where the frame for a journey has the following skeleton:

```
(JOURNEY
  (WITH SELF)
  (WITH POINT-OF-DEPARTURE)
  (WITH FINAL-DESTINATION)
  (WITH ROUTE)
  (WITH PATH))
```

Note how the point-of-departure of the new journey to be investigated in the FURTHER-INVESTIGATION frame is described as the city of a previous visit on the new route, called the-sub-route, which is described as a sub-route of the route we were on when this frame was instantiated.

Finally here is the frame for journey which contains references to all the previous method frames and combines them in a global method to find a path from one city to another one based on a depth first search. As usual comments between the lines are preceeded by ';;'.

```
(JOURNEY
 (WITH SELF)
 (WITH POINT-OF-DEPARTURE (THE FOCUS OF AN ATTENTION))
 (WITH FINAL-DESTINATION)
 (WITH ROUTE)
 (WITH PATH
    (WHEN THE-POINT-OF-DEPARTURE IS
          ;; we collect the possibilities
       ((THE POINT-OF-DEPARTURE OF A POSSIBLE-CONNECTIONS
             (WITH FIRST-POSSIBILITY (= THE-FIRST-POSSIBILITY))
             (WITH OTHER-POSSIBILITIES (= THE-OTHER-POSSIBILITIES)))
        (WHEN THE-FIRST-POSSIBILITY IS
             ;; we pick up the first possibility
          ((THE CONNECTION OF A POSSIBLE-CONNECTION
                (WITH DESTINATION (= THE-POSSIBLE-DESTINATION)))
           (IF-NOW THE-POSSIBLE-DESTINATION IS
```

```
((THE CITY OF A PREVIOUS-VISIT
      (WITH ROUTE (= THE-ROUTE)))
 ;; when the destination of this first possibility
 ;; occurs already on the path we know we are on a bad path
 ;; and therefore try out other possibilities
 (THE-POINT-OF-DEPARTURE IS
     (THE POINT-OF-DEPARTURE OF AN INVESTIGATION-OF-NEXT-POSSIBILITY
          (WITH OTHER-POSSIBILITIES (= THE-OTHER-POSSIBILITIES))
             (WITH FINAL-DESTINATION (= THE-FINAL-DESTINATION)))))
(ELSE
 ;; otherwise we continue the investigation
 (WHEN THE-POSSIBLE-DESTINATION IS
    ((= THE-FINAL-DESTINATION)
     ;; first case the destination of the possible-connection
     ;; is equal to the final destination
     ;;  we have found a possible route
     (THE-FIRST-POSSIBILITY IS
         (THE POSSIBLE-CONNECTION OF A POSSIBLE-PATH
              (WITH POINT-OF-DEPARTURE (= THE-POINT-OF-DEPARTURE))
              (WITH FINAL-DESTINATION (= THE-FINAL-DESTINATION)))))
   ((NOT (= THE-FINAL-DESTINATION))
     ;; second case the destination is NOT equal to the final one
     ;; we therefore investigate whether there is a path from this
     ;; possible-destination to the final one.
     (THE-POSSIBLE-DESTINATION IS
        (THE POSSIBLE-DESTINATION OF A FURTHER-INVESTIGATION
             (WITH FINAL-DESTINATION (= THE-FINAL-DESTINATION))
             (WITH POINT-OF-DEPARTURE (= THE-POINT-OF-DEPARTURE))
             (WITH PATH (= THE-PATH))
             (WITH ROUTE (= THE-ROUTE)))))
((THE POINT-OF-DEPARTURE OF A POSSIBLE-ROUTE
      (WITH FINAL-DESTINATION (= THE-FINAL-DESTINATION))
      (WITH CONNECTION (= THIS-CONNECTION)))
 ;; third case there is a possible journey from
 ;; this possible destination to the final one
 ;; in this case we are on the right track
 ;; and add the step from the point-of-departure
 ;; to the possible-destination to the path as
 ;; a whole
 (THE-FIRST-POSSIBILITY IS
     (THE POSSIBLE-CONNECTION OF A RECOMMENDED-PATH
          (WITH POINT-OF-DEPARTURE (= THE-POINT-OF-DEPARTURE))
          (WITH DESTINATION (= THE-POSSIBLE-DESTINATION))
          (WITH FINAL-DESTINATION (= THE-FINAL-DESTINATION))
          (WITH OTHER-CONNECTIONS (= THIS-CONNECTION)))))
;; If we know that there is no way
;; to get from this possible-destination to
;; the final one
((THE POINT-OF-DEPARTURE OF A BAD-CONNECTION
      (WITH FINAL-DESTINATION (= THE-FINAL-DESTINATION)))
 (THE-OTHER-POSSIBILITIES IS
      ;; we try out other possibilities
    (THE OTHER-POSSIBILITIES OF AN INVESTIGATION-OF-NEXT-POSSIBILIT
         (WITH FINAL-DESTINATION (= THE-FINAL-DESTINATION))
         (WITH POINT-OF-DEPARTURE
               (= THE-POINT-OF-DEPARTURE)))))))))))))
```

Let us now look at a particular example of reasoning based on these frames. The reader is advised to construct for himself a diagram of what happens here. First a description is sent to an object-expert saying that it is the path of a journey for going from London to Paris.

```
>> (TELL (AN OBJECT)
        (you-are-described-as
         (THE PATH OF A JOURNEY
              (WITH POINT-OF-DEPARTURE LONDON)
              (WITH  FINAL-DESTINATION PARIS))))
```

The journey frame is instantiated. LONDON-1 is now reasoning about London and PARIS-1 over Paris.

```
**
(JOURNEY (WITH SELF JOURNEY-1)
         (WITH POINT-OF-DEPARTURE LONDON-1)
         (WITH FINAL-DESTINATION PARIS-1)
         (WITH ROUTE ROUTE-1)
         (WITH PATH PATH-1))
**
(LONDON (WITH SELF LONDON-1))
```

London is described as the focus of an attention:

```
**
(ATTENTION
  (WITH SELF ATTENTION-1)
  (WITH FOCUS LONDON-1))
**
(PARIS (WITH SELF PARIS-1))
```

The connections become active:

```
**
(POSSIBLE-CONNECTIONS (WITH SELF POSSIBLE-CONNECTIONS-1)
                      (WITH POINT-OF-DEPARTURE LONDON-1)
                      (WITH FIRST-POSSIBILITY FIRST-POSSIBILITY-1)
                      (WITH OTHER-POSSIBILITIES OTHER-POSSIBILITIES-1))
```

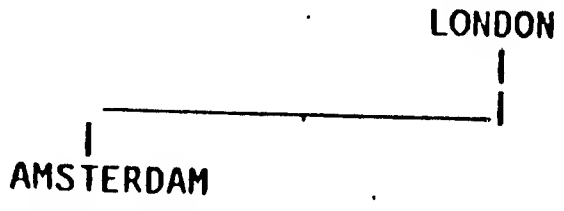
The first possibility is investigated:

```
**
(POSSIBLE-CONNECTION (WITH DESTINATION FIRST-POSSIBILITY-1)
                     (WITH POINT-OF-DEPARTURE LONDON-1)
                     (WITH DESTINATION AMSTERDAM-1))
```

It brings us to Amsterdam for which AMSTERDAM-1 is now responsible

```
**
(AMSTERDAM (WITH SELF AMSTERDAM-1))
```

The following part of the search tree has been investigated:



But Amsterdam is not Paris, i.e. PARIS-1 is not equal to AMSTERDAM-1, hence it is investigated whether you can go from Amsterdam to Paris

```
**
(FURTHER-INVESTIGATION (WITH SELF FURTHER-INVESTIGATION-1)
                       (WITH POINT-OF-DEPARTURE LONDON-1)
                       (WITH FINAL-DESTINATION PARIS-1)
                       (WITH ROUTE ROUTE-1)
                       (WITH PATH PATH-1)
                       (WITH POSSIBLE-DESTINATION AMSTERDAM-1))
```

This is done by a 'recursive instantiation' of the JOURNEY frame with Amsterdam the new point of departure:

```
**
(JOURNEY (WITH SELF JOURNEY-2)
         (WITH POINT-OF-DEPARTURE AMSTERDAM-1)
         (WITH FINAL-DESTINATION PARIS-1)
         (WITH ROUTE ROUTE-2)
         (WITH PATH PATH-1))
```

The reasoner makes records of what places have been visited:

```
**
(PREVIOUS-VISIT (WITH CITY AMSTERDAM-1) (WITH ROUTE ROUTE-2))
```

Amsterdam is now the focus of an attention:

```
**
(ATTENTION (WITH SELF ATTENTION-2)
           (WITH FOCUS AMSTERDAM-1))
```

More records of the route being followed

```
**
(ROUTE (WITH SELF ROUTE-3)
       (WITH SUB-ROUTE ROUTE-2)
       (WITH SUPER-ROUTE ROUTE-1))
**
(PREVIOUS-VISIT (WITH SELF PREVIOUS-VISIT-1)
                (WITH CITY LONDON-1)
                (WITH ROUTE ROUTE-2))
```

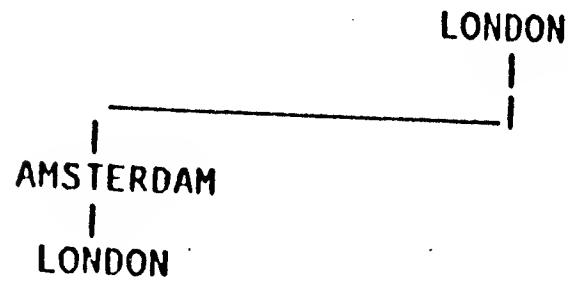
Here are the connections out of Amsterdam

```
**
(POSSIBLE-CONNECTIONS (WITH SELF POSSIBLE-CONNECTIONS-2)
                      (WITH POINT-OF-DEPARTURE AMSTERDAM-1)
                      (WITH FIRST-POSSIBILITY  FIRST-POSSIBILITY-2)
                      (WITH OTHER-POSSIBILITIES OTHER-POSSIBILITIES-2))
```

The first connection is investigated

```
**
(POSSIBLE-CONNECTION (WITH SELF FIRST-POSSIBILITY-2)
                     (WITH POINT-OF-DEPARTURE AMSTERDAM-1)
                     (WITH DESTINATION LONDON-1))
**
(EMPTY-LIST (WITH SELF OTHER-POSSIBILITIES-2))
```

The first connection brings the reasoner back to London (LONDON-1) so that the following portion of the tree has now come into view:



This is of course not the way to go, so the next possibility is investigated, i.e. OTHER-POSSIBILITIES-2

```
**
(INVESTIGATION-OF-NEXT-POSSIBILITY
        (WITH SELF INVESTIGATION-OF-NEXT-POSSIBILITY-1)
        (WITH OTHER-POSSIBILITIES OTHER-POSSIBILITIES-2)
        (WITH FINAL-DESTINATION PARIS-1)
        (WITH POINT-OF-DEPARTURE AMSTERDAM-1))
```

But this is the empty-list so that it is a bad way to go from Amsterdam in order to arrive at Paris:

```
**
(BAD-CONNECTION (WITH SELF BAD-CONNECTION-1)
                (WITH POINT-OF-DEPARTURE AMSTERDAM-1)
                (WITH FINAL-DESTINATION PARIS-1))
```

The reasoner backtracks to the next possibility with London the point-of- departure:

```
**
(INVESTIGATION-OF-NEXT-POSSIBILITY
 (WITH SELF INVESTIGATION-OF-NEXT-POSSIBILITY-2)
 (WITH OTHER-POSSIBILITIES OTHER-POSSIBILITIES-1)
 (WITH FINAL-DESTINATION PARIS-1)
 (WITH POINT-OF-DEPARTURE LONDON-1))
```

This is done by describing the other-possibilities as being under consideration:

```
**
(ATTENTION (WITH SELF ATTENTION-3)
           (WITH FOCUS OTHER-POSSIBILITIES-1))
```

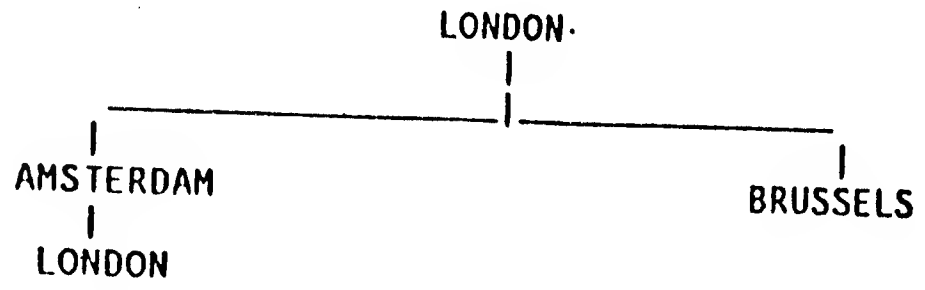
The connections are therefore propagated into the model:

```
**
(POSSIBLE-CONNECTIONS (WITH SELF POSSIBLE-CONNECTION-3)
                      (WITH POINT-OF-DEPARTURE LONDON-1)
                      (WITH FIRST-POSSIBILITY  FIRST-POSSIBILITY-3)
                      (WITH OTHER-POSSIBILITIES OTHER-POSSIBILITIES-3))
```

And the first one brings us to Brussels

```
**
(POSSIBLE-CONNECTION (WITH SELF FIRST-POSSIBILITY-3)
                     (WITH POINT-OF-DEPARTURE  LONDON-1)
                     (WITH DESTINATION BRUSSELS-1))
**
(BRUSSELS (WITH SELF BRUSSELS-1))
```

Here is the tree as developed so far



But Brussels is not Paris either, i.e. BRUSSELS-1 is not equal to PARIS-1. Hence it is investigated whether you can go from Brussels to Paris:

```
**
(FURTHER-INVESTIGATION (WITH SELF FURTHER-INVESTIGATION-2)
                       (WITH POINT-OF-DEPARTURE LONDON-1)
                       (WITH FINAL-DESTINATION  PARIS-1)
                       (WITH ROUTE ROUTE-1)
                       (WITH PATH PATH-1)
                       (WITH POSSIBLE-DESTINATION BRUSSELS-1))
```

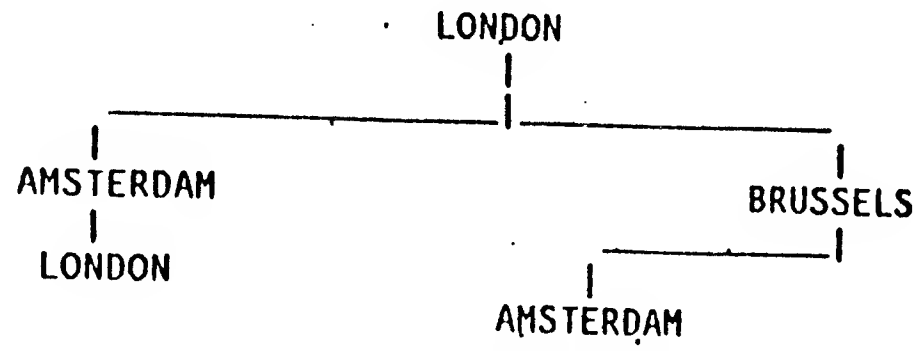
This causes another 'recursive instantiation' of the JOURNEY frame:

```
**
(JOURNEY (WITH SELF JOURNEY-3)
         (WITH POINT-OF-DEPARTURE BRUSSELS-1)
         (WITH FINAL-DESTINATION PARIS-1)
         (WITH ROUTE ROUTE-4)
         (WITH PATH PATH-1))
**
(PREVIOUS-VISIT (WITH SELF PREVIOUS-VISIT-2)
                (WITH CITY BRUSSELS-1)
                (WITH ROUTE ROUTE-4))
**
(ATTENTION (WITH SELF ATTENTION-4)
           (WITH FOCUS BRUSSELS-1))
**
(ROUTE (WITH SELF ROUTE-5)
       (WITH SUB-ROUTE ROUTE-4)
       (WITH SUPER-ROUTE ROUTE-1))
**
(PREVIOUS-VISIT (WITH SELF PREVIOUS-VISIT-3)
                (WITH CITY LONDON-1)
                (WITH ROUTE ROUTE-4))
```

Possibilities of connections departing from Brussels are now investigated:

```
**
(POSSIBLE-CONNECTIONS (WITH SELF POSSIBLE-CONNECTIONS-4)
                      (WITH POINT-OF-DEPARTURE BRUSSELS-1)
                      (WITH FIRST-POSSIBILITY FIRST-POSSIBILITY-4)
                      (WITH OTHER-POSSIBILITIES OTHER-POSSIBILITIES-4))
**
(POSSIBLE-CONNECTION (WITH SELF FIRST-POSSIBILITY-4)
                     (WITH POINT-OF-DEPARTURE BRUSSELS-1)
                     (WITH DESTINATION AMSTERDAM-1))
```

The first possibity brings us to Amsterdam (AMSTERDAM-1).



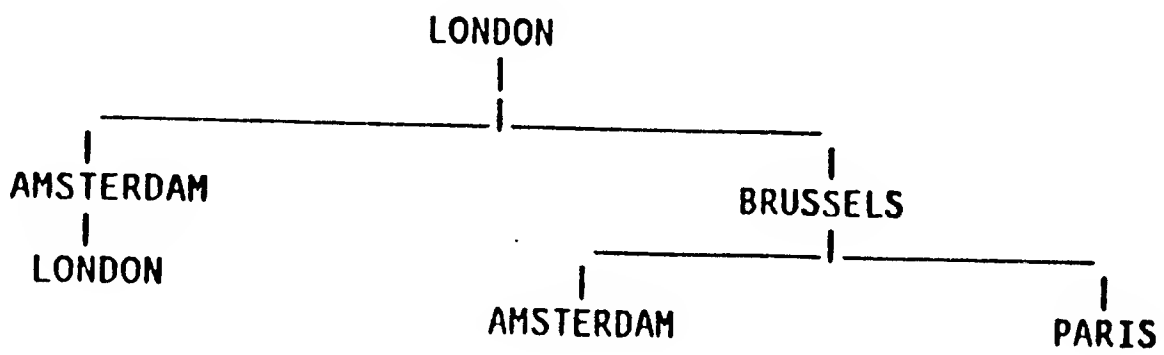
Amsterdam is not equal to London, so it is further investigated whether you can go from Amsterdam to Paris:

```
**
(FURTHER-INVESTIGATION (WITH POINT-OF-DEPARTURE BRUSSELS-1)
                       (WITH FINAL-DESTINATION  PARIS-1)
                       (WITH ROUTE ROUTE-4)
                       (WITH PATH PATH-1)
                       (WITH POSSIBLE-DESTINATION AMSTERDAM-1))
```

But the reasoner has figured out earlier on that it was bad to take a connection by way of Amsterdam for going from London to Paris. Hence the next possibility out of Brussels is investigated:

```
**
(INVESTIGATION-OF-NEXT-POSSIBILITY (WITH INVESTIGATION-OF-NEXT-POSSIBILITY-3)
                                   (WITH OTHER-POSSIBILITIES OTHER-POSSIBILITIES-4)
                                   (WITH FINAL-DESTINATION PARIS-1)
                                   (WITH POINT-OF-DEPARTURE BRUSSELS-1))
**
(ATTENTION (WITH SELF ATTENTION-5)
           (WITH FOCUS OTHER-POSSIBILITIES-4))
**
(POSSIBLE-CONNECTIONS (WITH SELF POSSIBLE-CONNECTIONS-5)
                      (WITH POINT-OF-DEPARTURE BRUSSELS-1)
                      (WITH FIRST-POSSIBILITY  FIRST-POSSIBILITY-5)
                      (WITH OTHER-POSSIBILITIES OTHER-POSSIBILITIES-5))
**
(POSSIBLE-CONNECTION (WITH CONNECTION FIRST-POSSIBILITY-5)
                     (WITH POINT-OF-DEPARTURE BRUSSELS-1)
                     (WITH DESTINATION PARIS-1))
**
(EMPTY-LIST (WITH SELF OTHER-POSSIBILITIES-5))
```

This possibility has Paris (PARIS-1) as final-destination. We end up with the following search tree:



That is where we have to be so a path is constructed that goes from Brussels (BRUSSELS-1) to Paris (PARIS-1).

```
**
(POSSIBLE-PATH (WITH SELF POSSIBLE-PATH-1)
               (WITH POSSIBLE-CONNECTION FIRST-POSSIBILITY-5)
               (WITH POINT-OF-DEPARTURE BRUSSELS-1)
**             (WITH FINAL-DESTINATION PARIS-1))
(POSSIBLE-ROUTE (WITH SELF POSSIBLE-ROUTE-1)
                (WITH CONNECTION FIRST-POSSIBILITY-5)
                (WITH POINT-OF-DEPARTURE BRUSSELS-1)
**              (WITH FINAL-DESTINATION PARIS-1))
(RECOMMENDED-CONNECTION (WITH SELF FIRST-POSSIBILITY-5)
                        (WITH POINT-OF-DEPARTURE BRUSSELS-1)
**                      (WITH DESTINATION PARIS-1))
(RECOMMENDED-PATH (WITH SELF RECOMMENDED-PATH-1)
                  (WITH POSSIBLE-CONNECTION FIRST-POSSIBILITY-3)
                  (WITH POINT-OF-DEPARTURE  LONDON-1)
                  (WITH FINAL-DESTINATION PARIS-1)
                  (WITH DESTINATION BRUSSELS-1)
                  (WITH OTHER-CONNECTIONS FIRST-POSSIBILITY-5))
```

This path is then completed by introducing a connection from London (LONDON-1) to PARIS (PARIS-1).

```
**
(COMBINATION-OF-CONNECTIONS (WITH SELF COMBINATION-OF-CONNECTIONS-1)
                            (WITH FIRST-CONNECTION  FIRST-POSSIBILITY-3)
**                          (WITH SECOND-CONNECTION FIRST-POSSIBILITY-5))
(RECOMMENDED-CONNECTION (WITH SELF FIRST-POSSIBILITY-3)
                        (WITH POINT-OF-DEPARTURE  LONDON-1)
**                      (WITH DESTINATION BRUSSELS-1))
(POSSIBLE-ROUTE (WITH SELF POSSIBLE-ROUTE-2)
                (WITH CONNECTION FIRST-POSSIBILITY-3)
                (WITH POINT-OF-DEPARTURE LONDON-1)
                (WITH FINAL-DESTINATION PARIS-1))
```

The job is done !

It might be of interest to know that this reasoning process required 258 facts, 19 experts and 51 rules. It is especially important to observe that portions of the network that are not relevant to the given problem were not developed, in particular BERLIN never came into the model.

Another interesting observations is that it is possible to develop a repertoire of general problem solving methods (like DEPTH-FIRST SEARCH, BREADTH-FIRST, etc.) by introducing method frames with a general scope of application and by introducing general frames for representing alternatives. In this way we gain back the generalizations which were seemingly thrown away when we introduced the explicit control of reasoning principle.

DISCUSSION

The music example was suggested by David Levitt. Levitt and the author are currently working on

an expert for tonal harmony which would accept high level descriptions of music and construct musical scores. We attempt to simulate esthetic reasoning which was also studied by Kahn (1979) for the domain of animation.
The idea to view a declarative language as a programming language has been proposed by Kowalski (1974).

7. COMMUNICATION

This chapter does not extend the reasoning model as such, instead we look at the issue of communication. Communication is a particular way of interacting with a model built up by the reasoner in order to see what results have been obtained or in order to cause certain problem solving behavior. Although this sort of interaction is not a topic of this work as such, we think it is necessary to give at least some idea of what is possible. We do this by constructing a relatively simple interface that can be used to converse with the reasoner. It will be clear how more complex interfaces, possibly culminating in a natural language conversation module, could be constructed.

This chapter contains two parts. In the first part we present a communication language. Then we give some examples of a discussion with the reasoner. This discussion will make use of the frames from the music-domain introduced in the previous chapter.

1. THE COMMUNICATION LANGUAGE

The communication language consists of a number of constructs for primitive speech acts: introducing, naming, predication, requesting and justifying.

PREDICATION

The most common speech act is the predication of a certain description to a particular object. The result of predication is that the object is described in terms of the description. Predication is represented as follows:

```
<description-of-object> IS <description-of-predicate>
```

as in

```
>> John is (the father of a family)
```

First an attempt will be made to find the referent of <description-of-object> if that referent is found the <description-of-predicate> will be predicated, which means according to the reasoning model developed in earlier chapters that a message is sent to the expert reasoning about the referent saying that it is described by this description. If no referent can be found, the <description-of-object> is sent to an anonymous object-expert which will start working out the details. This object-expert will also receive the <description-of-predicate> as one of its descriptions.

We make use of the methods discussed in the chapter on reasoning to find the referent of a description. In particular, we first try to recover the instantiation underlying the description. The referent of the description is the filler of the view (as long as this view is projective).

REQUESTING

All reasoning would be useless if we cannot extract the results. There are two types of

queries: informative and non-informative. Informative requests (also known as WH-questions) probe for a description of a particular object. Non-informative requests (also known as yes-no questions) probe whether a particular predication is true for a certain object.

The syntax for *non-informative* questions is

IS <description-of-object> <questioned-predication>?

Where <description-of-object> introduces a particular object and the <questioned-predication> is a certain description that must hold for the object. Response to this action can be of two forms:

YES,
<description-of-object> IS <questioned-predication>

or

I DON'T KNOW

which means that no description of the given form is known to be true for the object referred to by <description-of-object>. When instead there is a match with a negative description, the reasoner will respond with

NO,
<description-of-object> IS (NOT <questioned-predication>)

Here is an example:

```
>> Is John (the father of a family)?
YES,
JOHN IS (THE FATHER OF A FAMILY
                (WITH MOTHER MARY))
```

The same method is used as before to find the referent of the <description-of-object>. If that referent is found an attempt is performed to match the <questioned-predication> with a description in the expert of the referent. If no referent could be found, a new expert with the <description-of-object> as initial specification will be created and all consequents deduced. When that is done an attempt to match will be performed. The mechanism for matching is identical to the one used for matching a condition in a conditional description, as discussed in the chapter on reasoning. So the questioned-predication will be decomposed if necessary.

The answer to the question contains all descriptions which match with the request. If the request contains a partial description, it is completed by adding fillers for which an individual-concept is known. When there is more than one description that matches, indices are assigned to each description for ease of further reference.

The syntax for *informative requests* is

WHO IS <description-of-object>?

or

WHAT IS <description-of-object>?

depending whether the referent is a living thing or not. The reasoner will respond with a description of the <description-of-object>.
For example,

```
>> WHO IS (THE FATHER OF A FAMILY (WITH MOTHER MARY))?
JOHN IS (THE FATHER OF A FAMILY (WITH MOTHER MARY))
```

Again an attempt is made to find the referent of <description-of-object> and a new object-expert will be created if necessary. The problem is how the reasoner should talk back, i.e. how should a certain individual be introduced to the user. Here we take the following method. We look whether the individual is described with an individual-concept like JOHN, MARY, etc. If that is so an individual description will be made based on this concept. Otherwise we use (for the time being at least) the actual name of the expert.

JUSTIFYING

A reasoner that is unable to justify its reasons for believing a certain fact is dumb and dangerous. Because the reasoner keeps track of the dependencies between the facts. We can make use of these dependency structures to make the reasoner justify its deductions.

The following construct is used to ask for justifications:

```
(WHY <argument>)
```

where the argument is optional.
When no argument is given, the first description of the previous list of printed descriptions is assumed. When an argument is given, it is the index of the list of descriptions printed out before.

Execution of the WHY-act will cause the reasoner to print out the descriptions which were antecedents of the description referred to by the <argument>. The same conventions are used as those for a question. In other words, there will be a list of antecedents and indices are assigned so that one can refer to them later on.

COMMUNICATING FRAMES

At all times it is possible to introduce definitions for new concepts in terms of a frame for that concept.

Also it might be useful to look at the definition of a certain frame. This can be done by the following construct:

```
(GIVE-FRAME <frame-name>)
```

where <frame-name> is the name of a frame.

Execution of this action will cause the definition of the frame, as it was supplied by the user to be printed.

2. AN EXAMPLE

The best way to understand how this communication language operates is by studying an example. The following is an extract of a communication with the current implementation of the reasoner via an interpreter of this communication language. The communication is about music.

We start by asking whether C is the root of a half-step. The answer will be affirmative and the reasoner responds by enumerating the descriptions that confirm this answer:

```
>> Is C (the root of a half-step)?
YES,
C IS (THE ROOT OF A HALF-STEP (WITH END-TONE C-SHARP))
```

Note that the description has been completed by introducing the slot-filler of the end-tone. Note also that the request made use of a partial description. If there is more than one description matching this partial description they are all displayed.

The reasoner had to do a lot of things in order to answer this question. The object corresponding to the individual-concept C did not yet exist in the model. So it was created. This means also that all other tones in the tone structure are entered into the model and all half-step and whole-step relations. Once this was achieved the question could be asked.

Because of the multi-directionality of the descriptions, we can ask a similar question about C-SHARP:

```
>> Is C-Sharp (the end-tone of a half-step)?
YES,
C-SHARP IS (THE END-TONE OF A HALF-STEP (WITH ROOT C))
```

The reasoner did not have to do any work at all for this question, because all relevant objects and facts existed already in the model.

Now a question about whole-steps.

```
>> Is C (the root of a whole-step)?
YES,
C IS (THE ROOT OF A WHOLE-STEP (WITH END-TONE D))
```

Next a request that should result in a negative response:

```
>> IS C D?
NO,
C IS (NOT D)
```

We now try to solve an instance of the passing chord problem. Let us consider the one where the start-tone is C and the end-tone is D. We ask the following question and get two answers back:

```
>> WHAT IS (the passing-chord of a possible-passing-chord
                (with start-tone C)
                (with end-tone D))?
(THE PASSING-CHORD OF A POSSIBLE-PASSING-CHORD
     (WITH START-TONE C)
     (WITH END-TONE D))
IS
1. : G
2. : C-SHARP
```

In other words, G and C-SHARP are tones for a possible chord for going from C to D. We now illustrate the justification capacity.

First we ask for a justification of the first description:

```
>> (WHY)
(THE PASSING-CHORD OF A POSSIBLE-PASSING-CHORD
     (WITH START-TONE C)
     (WITH END-TONE D))
IS G
BECAUSE
  1. G IS (THE ROOT OF A FIFTH (WITH END-TONE D))
  2. C IS
       (IF G IS
           ((THE ROOT OF A FIFTH
                 (WITH END-TONE D))
            (THE END-TONE OF A POSSIBLE-PASSING-CHORD
                 (WITH PASSING-CHORD G)
                 (WITH START-TONE C))))
```

In other words G is a possible-passing-chord because it is the root of a fifth with end-tone D and because if there is this relation then the passing-chord relation holds. Let us now pursue the question why the fifth-relationship itself is justified.

```
>> (WHY)
G IS (THE ROOT OF A FIFTH (WITH END-TONE D))
BECAUSE
  1. : C IS (THE ROOT OF A WHOLE-STEP (WITH END-TONE D))
  2. : G IS (IF C IS
                ((THE ROOT OF A WHOLE-STEP (WITH END-TONE (= THE-FIFTH)))
                 (THE ROOT OF A FIFTH (WITH END-TONE (= THE-FIFTH)))))
```

So the presence of the fifth depends on two facts: C must be the root of a whole-step where D is the end-tone and there is a conditional description that says if this whole-step relation is there, then G can be described as the root of a fifth. We now investigate where this conditional description comes from:

```
>> (WHY 2)
G IS (IF C IS
         ((THE ROOT OF A WHOLE-STEP (WITH END-TONE (= THE-FIFTH)))
          (THE ROOT OF A FIFTH (WITH END-TONE (= THE-FIFTH)))))
BECAUSE
1. : G IS (THE ROOT OF A FOURTH (WITH END-TONE C))
2. : G IS (IF G IS
              ((THE ROOT OF A FOURTH
                    (WITH END-TONE (= THE-FOURTH)))
               (IF THE-FOURTH IS
                   ((THE ROOT OF A WHOLE-STEP (WITH END-TONE (= THE-FIFTH)))
                    (THE ROOT OF A FIFTH (WITH END-TONE (= THE-FIFTH)))))))
```

So the presence of the conditional depends on two facts: G must be the root of a fourth

whose end-tone is C and there is a conditional that says when you have a fourth, you get another conditional that will try to deduce another whole-step from this fourth. Let us investigate the justification for this fourth. We should see the following: the search for a half-step depending on the successful discovery of a whole-step. The search for this whole-step depending on the successful discovery of another whole-step. The search for this whole-step was caused by the find-fourth frame.

```
>> (WHY)
G IS (THE ROOT OF A FOURTH (WITH END-TONE C))
BECAUSE
1. : B IS (THE ROOT OF A HALF-STEP (WITH END-TONE C))
2. : G IS
       (IF B IS
           ((THE ROOT OF A HALF-STEP (WITH END-TONE (= THE-FOURTH)))
            (THE ROOT OF A FOURTH (WITH END-TONE (= THE-FOURTH)))))
```

This was the half-step.

```
>> (WHY 2)
G IS
  (IF B IS
      ((THE ROOT OF A HALF-STEP (WITH END-TONE (= THE-FOURTH)))
       (THE ROOT OF A FOURTH (WITH END-TONE (= THE-FOURTH)))))
BECAUSE
1. : A IS (THE ROOT OF A WHOLE-STEP (WITH END-TONE B))
2. : G IS
       (IF A IS
           ((THE ROOT OF A WHOLE-STEP (WITH END-TONE (= THE-SECOND-END-TONE)))
            (IF THE-SECOND-END-TONE IS
                  ((THE ROOT OF HALF-STEP (WITH END-TONE (= THE-FOURTH))
                   (THE ROOT OF A FOURTH (WITH END-TONE (= THE-FOURTH)))))))
```

This was the whole-step

```
>> (WHY 2)
G IS
(IF A IS
    ((THE ROOT OF A WHOLE-STEP (WITH END-TONE (= THE-SECOND-END-TONE)))
     (IF THE-SECOND-END-TONE IS
         ((THE ROOT OF HALF-STEP (WITH END-TONE (= THE-FOURTH))
              (THE ROOT OF A FOURTH (WITH END-TONE (= THE-FOURTH)))))))
BECAUSE
1. : G IS (THE ROOT OF A WHOLE-STEP (WITH END-TONE A))
2. : G IS (IF G IS
              ((THE ROOT OF A WHOLE-STEP
                    (WITH END-TONE (= THE-FIRST-END-TONE)))
               (IF THE-FIRST-END-TONE IS
                  ((THE ROOT OF A WHOLE-STEP
                        (WITH END-TONE (= THE-SECOND-END-TONE)))
                   (IF THE-SECOND-END-TONE IS
                         ((THE ROOT OF A HALF-STEP
                               (WITH END-TONE (= THE-FOURTH)))
                          (THE ROOT OF A FOURTH
                               (WITH END-TONE (= THE-FOURTH)))))))))
```

This was the other whole-step

```
>> (WHY 2)
G IS (IF G IS
         ((THE ROOT OF A WHOLE-STEP
               (WITH END-TONE (= THE-FIRST-END-TONE)))
          (IF THE-FIRST-END-TONE IS
              ((THE ROOT OF A WHOLE-STEP
                    (WITH END-TONE (= THE-SECOND-END-TONE)))
               (IF THE-SECOND-END-TONE IS
                   ((THE ROOT OF A HALF-STEP
                         (WITH END-TONE (= THE-FOURTH)))
                    (THE ROOT OF A FOURTH
                         (WITH END-TONE (= THE-FOURTH))))))))
BECAUSE
G IS (THE ROOT OF A FIND-FOURTH)
```

All this is justified by the fact that we are looking for a fourth.

The ultimate justification is the search for a passing-chord. So if we continue to ask for justifications the reasoner would eventually respond with:

```
BECAUSE
  YOU ASKED ME
```

We could ask many more questions and deduce other possible-passing-chord relationships. But these communications should give some idea about possible interactions with a reasoner. Other examples have of course been given in the introductory chapter.

DISCUSSION

There is an extensive literature on question-answering systems (see e.g. Lehnert,1978) and on flexible interaction with knowledge-based expert-systems (see e.g. Davis,1977).

Although the expressions used in the description language look somewhat like natural language the jump to natural language proper is still very large. In fact we believe that an adequate commnunication module has to be a reasoner itself. In Steels(1978) we make some specific proposals to go about doing this. In that paper we propose to view a grammar as a body of concepts and their definitions that is consulted by a reasoner to perform linguistic tasks like parsing and production. We are currently working on such a 'frame-based' conceptual grammar.

SUMMARY AND CONCLUSIONS

8. SUMMARY AND CONCLUSIONS

We have tried to explain how the human mind solves problems by reducing this question to a more general one: how is it possible for a physical system, i.e. a system constrained in time and space, to solve problems.

The results obtained from working on this more general problem are threefold: First of all we have been able to formulate a set of principles that every physical reasoning system must embody and we have given a rational justification for each of these principles. This rational justification is in the form of a functional argument that indicates that the principle is necessary if one wants to explain how a physical system is able to reason.

Second we have constructed a detailed model of a reasoning system which consists of a framework of concepts, systems and behaviors that specify in detail but still on a sufficiently abstract, i.e. implementation independent level, what a reasoning system might look like. One of the major parts of this framework is a description language that can be used to represent knowledge in a form suitable for explaining how reasoning works.

Finally we have constructed a concrete system that is based on the theoretical framework developed here and that has the capacity to perform certain types of reasoning. The details of this concrete system were not discussed at all because that would double the size of this document and is only of interest and understandable to the specialist anyway. Instead we extracted a large set of examples from actual interactions with this system.

There are numerous topics that need to be investigated further. Here are some of them.

(ii) REFINEMENTS OF THE REASONER

There are a number of obvious inadequacies that need to be taken care of. For example, it is well known that natural reasoning makes use of so-called default specifications, which are predications which are usually but not always true.(Minsky, 1974) We have designed and implemented a particular sort of truth maintenance system (similar to the one proposed in Doyle,1977) that is able to perform non-monotic reasoning as required to deal with defaults.

It is also well known that in order to reason adequately over changing worlds or belief structures it is necessary to have some sort of context-mechanism that partitions the model in several related sub-models. (See the discussion of the frame-problem in McCarthy and Hayes(1968) and specific context-mechanisms in Rulifson, et.al (1973), McDermott and Sussman (1974), or Fikes and Hendrix (1978)). We have been working on such a mechanism in the context of the present reasoner. Reports on our version of truth-maintenance and context handling will appear soon.

There are many other small points that need further investigation. For example it might be of interest to specify the cardinality of non-projective aspects. We should add some

SUMMARY AND CONCLUSIONS

primitive frames for doing arithmetic,etc.

(ii) CONCEPTUAL ANALYSIS.

Another thing that needs to be done is a large scale investigation into problems of conceptualization, eventually leading to a set of principles of conceptual analysis. Fortunately there has already been a lot of work in this area both by philosophers (see e.g. Reichenbach (1947), Carnap (1965) , and many others) and AI-researchers (see e.g. Martin (1978), Schank (1975), Hayes (1978), Wilks (1978),etc.).
Conceptual analysis is here interpreted in a broad sense. The study of problem solving concepts is also part of conceptual analysis.

(iii) OTHER TYPES OF REASONING.

Another thing that needs to be done is study other types of reasoning and see whether new representational tools and special mechanisms are needed. Some promising areas are

(i) Reasoning about knowledge currently under investigation by a number of researchers, such as McCarthy (1978), Moore (1979), a.o.

(ii) Reasoning by analogy, a domain which is also one of the areas which has had a lot of attention recently, see e.g. Moore and Newell (1973), Brown (1976), Winston (1978), a.o. Although the mechanisms of explicit control of reasoning might be a good first base to start investigating reasoning by analogy, it might be necessary to have special mapping mechanisms that project one frame onto another one.

(iii) Higher order reasoning. In Steels(1978) we made already concrete proposals in terms of a VIEWED-AS operator (which is essentially a restricted form of the Moore and Newell (1973) /-operator) in order to deal with higher order reasoning. We do not expect problems in mechanizing this proposal but it would require another extension.

(iv) Plausible reasoning. It should be further investigated whether probability or possibility estimates should be added to the predication of an assertion. Proposals for this can be found in Polya(1959).

etc.

(iii) MEMORY.

It is possible to gain enormously in efficiency by storing and re-invoking (partial) models in addition to the frames on which they are based. For example, in the music domain, we could store a model that contains basic objects and relations such as the tone structure with half-step and whole-step relations. Each time something needs to be done in the music world, this model could act as a first basis and initial instantiations would not have to be made each time anew. This raises the problem of how to invoke these partial models. Some recent work by Minsky (1979) on the so-called K-line theory contains ideas on how to proceed developing this area.

(iv) LEARNING.

Then there is the learning problem. We would like to study abstraction processes that

SUMMARY AND CONCLUSIONS

would allow transfer of knowledge from a model to a frame. We expect such a study to go along the lines of the accomodation mechanisms sketched in Piaget(1975). But learning implies also that one develops critics who can reject a new piece of information when it is not plausible, that domain-knowledge is revised during reasoning (which implies non-monotonicity), etc.

(V) COMMUNICATION.

Also we need to study more elaborate communication mechanisms so that it is possible to transmit the results obtained by the exploration of a model of a problem situation and that the exploration itself can be guided by conversation with another intelligence. The fact that we propose to use the reasoning system itself as a foundation for mechanizing linguistic processing is an interesting idea in itself that should lead to more robust and more complex natural language systems than the ones presently in operation.

(VI) IMPLEMENTATION

Finally we will be working on new implementations of the reasoner. The present version is written in Maclisp and has been implemented on the MIT-AI Lab PDP-10. A Lisp-machine implementation was completed in the summer of 1979.

REFERENCES

ANDERSON, John R., and BOWER, Gordon H.
1973 *Human Associative Memory.*
Washington, D.C.: V.H. Winston and Sons.

BAKER, Henry H. Jr.
1978 "Actor systems for real-time computation."
Technical Report 197. Cambridge MA: LCS, March 1978.

BOBROW, Daniel G., and COLLINS, Allan (eds.)
1975 *Representation and Understanding; Studies in Cognitive Science.*
New York: Academic Press.

BOBROW, Daniel G., and WINOGRAD, Terry.
1977 "An overview of KRL, a knowledge representation language."
Cognitive Science, Vol.1, No. 1, January 1977, pp 3-46.

BOBROW, Daniel G., and NORMAN, Donald A.
1975 "Some principles of memory schemata."
In *Representation and Understanding.* Daniel Bobrow and
Allan Collins, eds. New York: Academic Press, pp. 103-129.

BOBROW, Robert and BROWN John S.
1975 "Systematic understanding: synthesis, analysis and contingent
knowledge in specialized understanding systems."
In *Representation and Understanding.* Daniel Bobrow and
Allan Collins, eds. New York: Academic Press, pp. 103-129.

BORNING, Alan H.
1979 "Thinglab - A constraint-oriented simulation laboratory."
Ph.D. thesis draft. Stanford Ca: Stanford University, March, 1979.

BRACHMAN, Ronald J.
1978 "A structural paradigm for representing knowledge."
BBN Report No. 3605. Cambridge, MA: BBN Inc. May 1978.

CARNAP, Rudolf
1958 *Introduction To Symbolic Logic And Its Applications.*
New York: Dover Publications.

CHANG, C. and LEE, R.C.
1973 *Symbolic Logic and Mechanical Theorem Proving*
New York: Academic Press.

CHARNIAK, Eugene

1972 "Toward a model of children's story comprehension"
AI Technical Report no. 266. Cambridge MA: Artificial Intelligence Laboratory, MIT, December 1972.

CHARNIAK, Eugene
1977 "A framed PAINTING: the representation of a common sense knowledge fragment."
Cognitive Science, Vol 1, no 4, October, 1977, pp. 355-394.

CLIPPINGER, John H. Jr.
1977 *Meaning and Discourse: A computer Model of Psychoanalytic Speech and Cognition.*
Baltimore: the John Hopkins University Press.

DAVIES, D.J.M.:
1973 "Popler: A POP-2 PLANNER"
MIP-89. Edinburgh: School of AI, University of Edinburgh.

DAVIS, Randall:
1976 "Applications of meta level knowledge to the construction, maintenance and use of large knowledge bases.
Memo AIM-283. Stanford, Ca: Stanford AI Lab, July 1976.

DE KLEER, Johan, DOYLE, Jon, RICH, Charles, STEELE, Guy L., and SUSSMAN, Gerald J.
1977 "AMORD, a deductive procedure system"
Working Paper No 151. Cambridge, MA: Artificial Intelligence Laboratory, August 1977.

DE KLEER, Johan, DOYLE, Jon, RICH, Charles, STEELE, Guy L., and SUSSMAN, Gerald J.
1977 "Explicit control of reasoning."
AI Memo No. 427. Cambridge, MA: Artificial Intelligence Laboratory, June 1977.

DE KLEER, Johan and SUSSMAN, Gerald J.
1978 "Propagation of constraints applied to circuit synthesis"
AI Memo No 485. Cambridge, MA: Artificial Intelligence Laboratory, September 1978.

DOYLE, Jon.
1976 "Analysis by propagation of constraints in elementary geometry problem solving."
AI Working Paper No 108. Cambridge, Ma: Artificial Intelligence Laboratory, June 1976.

DOYLE, Jon.
1976 "The use of dependency relationsips in the control of reasoning"
AI Working Paper No 133. Cambridge, MA: Artificial Intelligence Laboratory,

November 1976.

ERNST, George W. and NEWELL, Allen.
1969 *GPS: A Case Study In Generality and Problem-solving.*
New York: Academic Press.

FAHLMAN, Scott.
1973 "A hypothesis-frame system for recognition problems."
Working Paper. No 57. Cambridge Ma: Artificial Intelligence Laboratory, December 1973.

FAHLMAN, Scott.
1977 "A system for representing and using real-world knowledge."
Ph.D. thesis draft. Cambridge Ma: Artificial Intelligence Laboratory, June 1977.

FENNELL, R.D. and LESSER, V.R.
1975 "Parallelism in AI problem solving. A case study of Hearsay II."
Technical Report. Pittsburgh, PA: Dept of Computer Science, Carnegie-Mellon University, October 1975.

FIKES, Richard, C.
1976 "The deduction component."
In *Speech Understanding Research*
Donald E. Walker, ed. Final Technical Report No 4762. Menlo Park, Ca: Stanford Research Institute, October 1976.

FIKES, Robert and HENDRIX, Gary.
1977 "A network-based knowledge representation and its natural deduction system."
In *Proceedings of the Fifth International Joint Conference on Artificial Intelligence*, Cambridge Ma, August 1977, pp. 235 - 246.

FIKES, R.E. and NILLSON, N.J.
1971 "STRIPS: A new approach to the application of theorem proving to problem solving."
Artificial Intelligence, 2, 1971, pp. 189 - 208.

FREUDER, Eugene C.
1976 "Synthesizing constraint expressions."
AI Memo No 370. Cambridge, Ma: Artificial Intelligence Laboratory, July 1976.

GELERNTER, Herbert.
1962 "Machine-Generated Problem-solving Graphs."
In *Mathematical Theory of Automata.* pp. 179 - 203.

GOLDSTEIN, Ira P. and MILLER, Mark L.
1976 "Structured planning and debugging; A linguistic theory of design."
AI Memo No 387. Cambridge, Ma: Artificial Intelligence Laboratory, December 1976.

GOLDSTEIN, Ira P. and ROBERTS, K. BRUCE
1977 "The FRL Mannual"
AI Memo No. 409. Cambridge, Ma: Artificial Intelligence Laboratory.

HAWKINSON, L.
1979 "LMS-manual" Mimeo.

HAYES, Pat J.
1974 "Some problems and non-problems in representation theory."
In *Proceedings of AISB Conference* Sussex, July 1974.

HAYES, Pat J.
1977 "In defence of logic"
In *Proceedings of the Fifth International Joint Conference on Artificial Intelligence*, Cambridge, Ma. August 1977. pp. 559 - 565.

HAYES, Pat J.
1977 "The logic of frames" Mimeo.
Colchester: Dept of Computer Science, University of Essex. November 1977.

HAYES, Philip, J.
1975 "A representation for Robot Plans"
IJCAI 4. September 1975, pp. 181- 188.

HENDRIX, Gary G.
1975 "Partitioned networks for the mathematical modeling of natural language semantics."
Technical Report NL-28. Austin, Texas: Dept of Computer Science, The university of Texas at Austin, December 1975.

HEWITT, Carl.
1969 "PLANNER: A language for manipulating models and proving theorems in a robot."
IJCAI-69. Washington D.C.

HEWITT, Carl.
1971 "Procedural embedding of knowledge in PLANNER"
IJCAI-71. London.

HEWITT, Carl.
1972 "Description and theoretical analysis (using schemata) of PLANNER:

A language for proving theorems and manipulating models in a robot."
Cambridge, Ma: Artificial Intelligence Laboratory, April 1972.

HEWITT, Carl, BISHOP, P. and STEIGER, Richard.
1973 "A universal modular actor formalism for artificial intelligence"
In IJCAI-73. pp. 235-245.

HEWITT, Carl
1974 "Protection and Synchronization in actor systems"
Working Paper No 83. Cambridge, Ma: Artificial Intelligence Laboratory, November 1974.

HEWITT, Carl
1975 "How to use what you know"
Working Paper No 93. Cambridge, Ma: Artificial Intelligence Laboratory, May 1975.

HEWITT, Carl
1976 "Viewing control structures as patterns of passing messages"
AI Memo No 410. Cambridge, Ma: Artificial Intelligence Laboratory, December 1976.

HEWITT, Carl
1979 "Concurrent systems need both sequences and serializers."
Working paper No. 179. Cambridge Ma: Artificial Intelligence Laboratory, February 1979.

HEWITT, Carl, ATTARDI, Giuseppe and LIEBERMAN, Henry.
1978 "Specifying and proving properties of guardians for distributed systems"
Working Paper No. 172. Cambridge, Ma: Artificial Intelligence Laboratory, October 1978.

HEWITT, Carl and BAKER, Henry.
1977 "Laws for communicating parallel processes."
AI Working Paper. No. 134a. Cambridge, Ma: Artificial Intelligence Laboratory, May 1977.

HEWITT, Carl and SMITH, Brian.
1975 "Towards a programming apprentice"
IEEE Transactions on software engineering. SE-1,1.
March 1975, pp. 26-45.

HOLLERBACH, John. M.
1975 "Hierarchical shape description of objects by selection and modification of prototypes."
Technical Report No 346. Cambridge, Ma: Artificial Intelligence Laboratory.
November 1975.

JEFFERY, Mark J.
1978 "Representing "Place" in a frame system."
M.S. thesis, Cambridge, Ma: Artificial Intelligence Laboratory, January 1978.

KAHN, Kenneth M.
1975 "Mechanization of temporal knowledge"
MAC Technical Report, No. 155.
Cambridge, Ma: MIT, September 1975.

KAHN, Kenneth M.
1979 "Creation of computer animation from story description"
Ph.D. Thesis, Cambridge, Ma: Artificial Intelligence Laboratory.

KAY, Alan.
1972. "A personal computer for children of all ages."
In *Proceedings of the ACM National Conference* August 1972

KORNFELD, William A.
1979 "Using parallel processing for problem solving"
M.S. thesis, Cambridge, Ma: Artificial Intelligence Laboratory.

KUIPERS, Benjamin J.
1975 "A frame for frames: Representing knowledge for recognition."
In *Representation and understanding*
Daniel Bobrow and Alan Collins, eds. New York: Academic Press, p. 151 - 184.

KUIPERS, Benjamin J.
1977 "Representing Knowledge of large-scale space."
AI TR No 418. Cambridge, Ma: Artificial Intelligence Laboratory, July 1977.

LEVESQUE, H.J.
1977 "A procedural approach to semantic networks."
Technical Report. Toronto Department of Computer Science, University of Toronto.

MARTIN, William A.
1979 "Philosophical foundations for a linguistically oriented semantic network." Draft.

McCARTHY, John.
1959 "Programs with common sense"
In *Mechanization of Thought Processes*. Marvin Minsky, et.al.
London: Her Majesty's Stationery Office.

McCARTHY, John.
1977 "Epistemological Problems of Artificial Intelligence"
In *IJCAI-77* PP. 1038-1044.

McCARTHY, John and Pat HAYES
1969 "Some philosophical problems from the standpoint of Artificial Intelligence."
In *Machine Intelligence*, Vol 4. pp 463-502.

McDERMOTT, Drew V.
1975 "Very large Planner-type data bases."
Al Memo. No 339. Cambridge, Ma: Artificial Intelligence Laboratory.

McDERMOTT, Drew, V. and DOYLE, Jon.
1978 "Non-monotic logic I"
Al Memo No 486. Cambridge, Ma: Artificial Intelligence Laboratory.

McDERMOTT, Drew V. and SUSSMAN, Gerald J.
1974 "The Conniver Reference Manual"
Al Memo No 259a. Cambridge, Ma: Artificial Intelligence Laboratory.

MINSKY , Marvin L.
1959 "Some methods of Artifical Intelligence and Heuristic Programming."
In *Mechanization of Thought Processes*, Minsky, et.al.

MINSKY, Marvin L.
1961 "Steps toward Artificial Intelligence"
In *Proceedings of Institute of Radio Engineers*, January 1961.

MINSKY, Marvin L. (ed.)
1968 "Semantic Information Processing"
Cambridge, Ma: the MIT Press.

MINSKY, Marvin L.
1972 *Computation: Finite and Infinite Machines*. London: Prentice Hall.

MINSKY, Marvin L.
1974 "A framework for representing knowledge."
Al memo No 306. Cambridge Ma: Artifical Intelligence Laboratory.

MINSKY, Marvin, and PAPERT, Seymour.
1971 "Artificial Intelligence Progress Report"
In *Project MAC progress report VIII* July 1970 to July 1971.
pp. 129-224.

MINSKY, Marvin. L., et.al.
1959 *Mechanization of Thought Proceses*. Proceedings of a Symposium

held at the National Physical Laboratory on 24th, 25th, 26th and 27th November 1958. Vol. 1.
London: Her Majesty's Stationery Office.

MOORE, J. and NEWELL, A.
1973 "How can Merlin understand? "
Pittsburgh, Pa: Dept of Computer Science, Carnegie-Mellon University.

MOORE, Robert C.
1975 "Reasoning from incomplete knowledge in a procedural deduction system"
AI Technical Report No 347. Cambridge, Ma: Artificial Laboratory.

NEVINS, Arthur J.
1974 "Plane geometry theorem proving using forward chaining."
Cambridge, Ma: Artificial Intelligence Laboratory.

NEWELL, A. and SIMON, H.A.
1972 *Human Problem Solving* Englewood, Cliffs, N.J.: Prentice Hall.

NILSSON, Nils J.
1965 *Learning Machines; Foundations of Trainable Pattern-classifying systems*
New York: McGraw-Hill Book Cy.

NILSSON, Nils J.
1971 *Problem Solving Methods in Artificial Intelligence*
New York: McGraw-Hill Book Cy.

NORMAN, Donald A., RUMELHART, David E. and the LNR Research Group.
1975 *Explorations in Cognition*
San Francisco: W.H. Freeman and Cy.

OGDEN, C.K.
1968 *Basic English International Second Language*
New York: Harcourt, Brace and World.

PIAGET, Jean
1967 *Biologie et Connaissance*
Paris: Gallimard.

QUILLIAN, M. Ross
1968 "Semantic Memory".
In *Semantic Information Processing*, Minsky, Marvin (ed)
Cambridge, Ma: MIT Press, pp 227-270.

RIEGER, Chuck

1975 "One system for two tasks: A common sense algorithm memory that solves problems and comprehends language." Working paper No 114. Cambridge, Ma: Artifiical Intelligence Laboratory.

ROBERTS, R. Bruce and GOLDSTEIN, Ira P.
1977 "The FRL primer"
AI Memo No 408. Cambridge, Ma: Artificial Intelligence Laboratory.

RULIFSON, J.F., DERKSEN, J.A., and WALDINGER, Richard J.
1972 "QA4, a procedural calculus for intuitive reasoning"
Technical Note No 73. Menlo Park, Ca: Stanford Research Institute. Artificial Intelligence Center.

SACERDOTI, Earl D.
1977 "The nonlinear nature of plans"
In *IJCAI-1977*, Cambridge Ma., pp 206-214.

SCHANK, Roger C. and COLBY, Kenneth M. (eds)
1973 *Computer models of thought and language.*
San Francisco: W.H. Freeman and Cy.

SCHANK, Roger C. (ed)
1975 *Conceptual Information Processing*
Amsterdam, North-Holland.

SCRAGG, G.W.
1975 "Frames, planes and nets: a synthesis"
Working Paper No 19. Geneva, Institute for Semantics and Cognitive Studies.

SIMMONS, Robert F.
1973 "Semantic networks, their computation and use for understanding English sentences."
In *Computer Models of Thought and Language*. Roger C. Schank and Kenneth M. Colby (eds) San Francisco: W.H. Freeman and Cy. pp 63-113

SIMON, Herbert A.
1969 *The sciences of the Artificial*
Cambridge, Ma: MIT Press.

SMITH, Brian C.
1978 "A proposal for a computational model of anatomical and physiological reasoning." AI Memo No 493. Cambridge, Ma: Artificial Intelligence Lab.

SRINIVASAN, Chitoor V.
1976 "The architecture of coherent information systems: A general

problem solving system."
IEEE Transactions on computers. Vol. c-25, no.4, pp. 390-402.

STALLMAN, Richard M. and SUSSMAN, Gerald J.
1976 "Forward reasoning and dependency-directed backtracking in a system for computer-aided circuit analysis.
AI-memo 380. Cambridge, Ma: Artificial Intelligence Lab.

STANSFIELD, James L.
1975 "Programming a dialogue teaching situation."
Ph.D. thesis. Edinburgh: University of Edinburgh.

STEELE, Guy L. and SUSSMAN, Gerald J.
1978 "Constraints"
AI memo No 502. Cambridge, Ma: Artificial Intelligence Lab.

SUSSMAN, Gerald J. and McDERMOTT, Drew V.
1972 "From PLANNER to CONNIVER - a genetic approach."
In *Proceedings of the Fall Joint Computer Conference*, pp 1171-1179.

SUSSMAN, Gerald J. and McDERMOTT, Drew V.
1972 "Why coniving is better than planning."
AI Memo No. 255a. Cambridge, Ma: Artificial Intelligence Laboratory.

SUSSMAN, Gerald J., WINOGRAD, Terry and CHARNIAK, Eugene.
1971 "Micro-planner reference manual"
AI Memo No 203a. Cambridge, Ma: Artificial Intelligence Laboratory.

WILKS, Yorick
1972 *Grammar, Meaning and the machine analysis of language.*
London: Routledge.

WILKS, Yorick
1976 "Frames, stories, scripts and fantasies."
In *Coling*, Ottawa.

WILKS, Yorick
1977 "Language boundaries and knowledge structures"
In *International workshop on the cognitive viewpoint.*
Ghent, pp. 178-186.

WINOGRAD, Terry
1972 *Understanding Natural Language*
New York: Academic Press.

WINOGRAD, Terry
1975 "Frame representations and the declarative/procedural controversy."